МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
"МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ"»
ФАКУЛЬТЕТ КОМП'ЮТЕРНО-ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Вовкотруб Олександр Віталійович

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка блокчейн застосунку «Система приватного відновлення доступу до криптогаманців на основі доказів з нульовим розголошенням»

Шифр групи: ІК-9-21-Б1ІПЗ(4.0д)

Спеціальність: Інженерія програмного забезпечення

Робота на здобуття освітньо-кваліфікаційного рівня бакалавр

Науковий керівник _____ канд. техн. наук

(підпис) Гордієнко Олександр Миколайович

Допущено до захисту ДЕК

Завідувач кафедри _____ професор, док. екон. наук

(підпис) Кавун С.В.

Київ 2026

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО
«ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»»
ФАКУЛЬТЕТ КОМП'ЮТЕРНО-ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Рівень вищої освіти: перший (бакалаврський)

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

«Затверджую»
Завідувач кафедри
_____ Кавун С.В.
«___» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Вовкотруба Олександра Віталійовича

1. Тема кваліфікаційної роботи: Розробка блокчейн застосунку «Система приватного відновлення доступу до криптогаманців на основі доказів з нульовим розголошенням»

Керівник кваліфікаційної роботи: канд. техн. наук Гордієнко Олександр Миколайович

2. Строк подання студентом кваліфікаційної роботи на попередній захист: «___» _____ 2026 р.

3. Вихідні дані до кваліфікаційної роботи:

Стандарти: EIP-7579, EIP-4337, ДСТУ 3008:2015. Технології: Solidity 0.8.x, Noir/Aztec.nr, Go 1.21+, Next.js 15, React 19, Foundry, Turborepo. Інфраструктура: Aztec Network (devnet), Ethereum Sepolia, Wormhole Protocol. Платформа: Safe Smart Account (модульна архітектура).

4. Необхідний графічний та аналітичний матеріал:

Діаграми архітектури системи, порівняльні таблиці існуючих рішень, діаграми потоків даних, діаграми моделі загроз, таблиці газових витрат та продуктивності.

5. Календарний план виконання кваліфікаційної роботи:

№	Назва етапів	Планові строки
1	Аналіз предметної області та постановка задачі	01.02–15.02.2026
2	Огляд існуючих рішень та формулювання вимог	16.02–28.02.2026
3	Проєктування архітектури системи	01.03–15.03.2026
4	Реалізація смарт-контрактів (Solidity + Noir)	16.03–05.04.2026
5	Реалізація Relayer та Frontend	06.04–20.04.2026
6	Тестування та розгортання на тестовій мережі	21.04–05.05.2026
7	Аналіз безпеки та оформлення роботи	06.05–20.05.2026
8	Підготовка до захисту	21.05–31.05.2026

6. Дата видачі завдання: «___» _____ 2026 р.

Студент _____ Вовкотруб О.В.

Науковий керівник _____ Гордієнко О.М.

АНОТАЦІЯ

Вовкотруб Олександр Віталійович. «Система приватного відновлення доступу до криптогаманців на основі доказів з нульовим розголошенням» – Рукопис.

Кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» – Факультет комп'ютерно-інформаційних технологій ПрАТ «ВНЗ «МАУП» – Київ, 2026.

Актуальність теми дослідження полягає в тому, що втрати криптоактивів через загублені приватні ключі становлять мільярди доларів (2.3–3.7 мільйонів BTC безповоротно втрачені), а компрометація ключів склала 43.8% від загальних крадіжок у 2024 році. Існуючі рішення соціального відновлення не забезпечують приватності опікунів, що створює вектори для соціальної інженерії та підкупу.

Мета роботи полягає в розробці системи приватного соціального відновлення для Safe гаманців, що забезпечує приватність опікунів через zero-knowledge proofs, анонімне голосування, крос-чейн сумісність та інтеграцію як модуль Safe без модифікації ядра.

У роботі були використані такі методи дослідження як аналіз існуючих рішень, проєктування архітектури розподілених систем, розробка смарт-контрактів на Solidity та Noir, застосування zero-knowledge proofs для забезпечення приватності, тестування на тестових мережах Sepolia та Aztec Devnet.

Ключові слова: блокчейн, криптовалютний гаманець, соціальне відновлення, zero-knowledge proof, приватність, опікун, смарт-контракт, Aztec Network, Safe, Wormhole, крос-чейн, Noir, nullifier.

ANNOTATION

Vovkotrub Oleksandr Vitaliyovych. "System for Private Recovery of Access to Crypto Wallets Based on Zero-Knowledge Proofs" – Manuscript.

Bachelor's qualification work – Faculty of Computer and Information Technologies, Private Joint-Stock Company "Higher Educational Institution 'Interregional Academy of Personnel Management'" – Kyiv, 2026.

The urgency of the research topic is that losses of crypto assets due to lost private keys amount to billions of dollars (2.3–3.7 million BTC permanently lost), and private key compromise accounted for 43.8% of total thefts in 2024. Existing social recovery solutions do not ensure guardian privacy, creating vectors for social engineering and bribery.

The purpose of the work is to develop a private social recovery system for Safe wallets that ensures guardian privacy through zero-knowledge proofs, anonymous voting, cross-chain compatibility, and integration as a Safe module without modifying the core.

The following research methods were used: analysis of existing solutions, design of distributed system architecture, development of smart contracts in Solidity and Noir, application of zero-knowledge proofs for privacy, testing on Sepolia and Aztec Devnet test networks.

Key words: blockchain, cryptocurrency wallet, social recovery, zero-knowledge proof, privacy, guardian, smart contract, Aztec Network, Safe, Wormhole, cross-chain, Noir, nullifier.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ І СКОРОЧЕНЬ

ABI	—	Application Binary Interface (бінарний інтерфейс застосунку)
ACIR	—	Abstract Circuit Intermediate Representation (абстрактне проміжне представлення схем)
API	—	Application Programming Interface (програмний інтерфейс застосунку)
BTC	—	Bitcoin (криптовалюта Біткоїн)
CEI	—	Checks-Effects-Interactions (патерн безпеки смарт-контрактів)
CLI	—	Command Line Interface (інтерфейс командного рядка)
DAO	—	Decentralized Autonomous Organization (децентралізована автономна організація)
DApp	—	Decentralized Application (децентралізований застосунок)
DSL	—	Domain-Specific Language (предметно-орієнтована мова)
E2E	—	End-to-End (наскрізне тестування)
EIP	—	Ethereum Improvement Proposal (пропозиція покращення Ethereum)
EOA	—	Externally Owned Account (зовнішній акаунт)
ERC	—	Ethereum Request for Comments (стандарт Ethereum)
ETH	—	Ether (криптовалюта Ефір, нативний токен мережі Ethereum)
EVM	—	Ethereum Virtual Machine (віртуальна машина Ethereum)
gRPC	—	Google Remote Procedure Call (протокол віддаленого виклику процедур)
JSON	—	JavaScript Object Notation (текстовий формат обміну даними)
L1	—	Layer 1 (базовий рівень блокчейну, наприклад Ethereum mainnet)
L2	—	Layer 2 (рішення другого рівня масштабування)
MPC	—	Multi-Party Computation (багатостороннє обчислення)
NFT	—	Non-Fungible Token (невзаємозамінний токен)

P2P	—	Peer-to-Peer (однорангова мережа)
RPC	—	Remote Procedure Call (віддалений виклик процедур)
SDK	—	Software Development Kit (набір інструментів розробника)
TX	—	Transaction (транзакція в блокчейні)
UML	—	Unified Modeling Language (уніфікована мова моделювання)
UTXO	—	Unspent Transaction Output (невитрачений вихід транзакції)
VAA	—	Verified Action Approval (верифіковане схвалення дії, примітив протоколу Wormhole)
ZK	—	Zero-Knowledge (нульове знання / нульове розголошення)
ZKP	—	Zero-Knowledge Proof (доказ з нульовим розголошенням)
ZK-SNARK	—	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (стислий неінтерактивний аргумент знання з нульовим розголошенням)

ВСТУП

Криптовалютні гаманці базуються на асиметричній криптографії, де приватний ключ є єдиним засобом контролю над активами. Втрата приватного ключа або seed-фрази означає безповоротну втрату доступу до коштів, оскільки блокчейн-системи не мають центрального органу для відновлення доступу. За оцінками дослідників, від 2.3 до 3.7 мільйонів BTC є безповоротно втраченими через загублені приватні ключі, а компрометація приватних ключів склала 43.8% від загальних втрат криптоактивів у 2024 році.

Соціальне відновлення (social recovery) є перспективним рішенням цієї проблеми, проте традиційні реалізації зберігають адреси опікунів публічно на блокчейні, створюючи вектори для соціальної інженерії, підкупу та примусу. Існуючі комерційні рішення (Argent, Loopring, Safe) не забезпечують приватності опікунів, а академічні дослідження ZK-підходів залишаються теоретичними.

Актуальність теми зумовлена зростаючими втратами криптоактивів через загублені ключі та відсутністю production-ready рішення для приватного соціального відновлення з використанням zero-knowledge proofs.

Мета роботи — розробити систему приватного соціального відновлення для Safe гаманців, що забезпечує приватність опікунів, анонімне голосування, крос-чейн сумісність та інтеграцію як модуль Safe без модифікації ядра.

Об’єкт дослідження — процес відновлення доступу до криптовалютних гаманців.

Предмет дослідження — методи забезпечення приватності опікунів у системах соціального відновлення на основі доказів з нульовим розголошенням.

Методи дослідження: аналіз існуючих рішень, проєктування архітектури розподілених систем, розробка смарт-контрактів на Solidity та Noir, застосування zero-knowledge proofs для забезпечення приватності, тестування на тестових мережах.

Практичне значення полягає у створенні діючої системи, що дозволяє користувачам Safe гаманців налаштувати приватне соціальне відновлення з прихованими адресами опікунів та анонімним голосуванням. Система розгорнута на тестовій мережі Sepolia та Aztec Devnet.

Структура роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та чотирьох додатків.

У першому розділі проаналізовано проблему відновлення доступу до криптогаманців на реальних прикладах, розглянуто існуючі рішення, визначено цільову аудиторію та сформульовано задачу.

У другому розділі описано архітектуру системи з використанням UML-діаграм, представлено workflow налаштування та відновлення, описано внесок автора та ключові аспекти реалізації.

У третьому розділі проведено аналіз безпеки та приватності, описано методологію тестування та наведено результати оцінювання продуктивності.

У додатках наведено повні коди смарт-контрактів на Solidity (Додаток А), Noir (Додаток Б), Relayer-сервісу на Go (Додаток В) та демонстрацію роботи системи (Додаток Г).

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Проблема відновлення доступу до криптовалютних гаманців

1.1.1 Як працюють криптовалютні гаманці

Криптовалютний гаманець — це програма або пристрій, що зберігає приватний ключ користувача. Приватний ключ — це унікальне число (256-бітне), яке є єдиним засобом контролю над активами на блокчейні. На відміну від банківського рахунку, де банк може скинути пароль або підтвердити особу клієнта, у блокчейні немає центрального органу. Хто має приватний ключ — той контролює кошти [1].

Приватний ключ зазвичай представлений у вигляді seed-фрази — набору з 12 або 24 англійських слів (наприклад: “abandon ability able about above absent…”). Ця фраза дозволяє відновити ключ на будь-якому пристрої, але її втрата означає повну та безповоротну втрату доступу до активів.

1.1.2 Масштаб проблеми: реальні випадки

Проблема втрати доступу до криптовалют не є теоретичною — вона торкнулася мільйонів людей та мільярдів доларів:

Випадок Джеймса Хоуеллса (2013). Британський IT-інженер Джеймс Хоуеллс випадково викинув жорсткий диск із приватним ключем до 8 000 BTC. За поточним курсом це понад $200 мільйонів. Диск знаходиться на сміттєзвалищі в Ньюпорті (Уельс), і вже понад 10 років Хоуеллс намагається отримати дозвіл на розкопки [41].

Випадок Стефана Томаса (2011). Програміст Стефан Томас забув пароль до апаратного гаманця IronKey, на якому зберігалися 7 002 BTC (~$175 мільйонів). Пристрій дозволяє лише 10 спроб введення пароля, після чого дані знищуються назавжди. На момент публікації залишалося 2 спроби [42].

Загальна статистика. За оцінками дослідників, від 2.3 до 3.7 мільйонів BTC (приблизно 11-18% від максимальної емісії у 21 мільйон) є безповоротно втраченими через загублені приватні ключі [2]. З урахуванням видобутих

~19.8 мільйонів BTC, реально доступна пропозиція становить лише 15.8-17.5 мільйонів BTC [2].

Окрім втрат через загублені ключі, значні суми викрадаються через компрометацію приватних ключів. У 2024 році було викрадено $2.2 мільярди криптоактивів через 303 інциденти злому, що на 21.07% більше порівняно з попереднім роком [3]. Компрометація приватних ключів склала найбільшу частку викрадених коштів — 43.8% від загальних втрат у 2024 році [3].

1.1.3 Єдина точка відмови

Традиційні криптогаманці створюють критичну вразливість — єдину точку відмови (single point of failure). Як зазначає Віталік Бутерін, seed-фрази створюють нові вектори крадіжки: викрадення апаратного гаманця або seed-фрази надає повний доступ до активів. Це фундаментальна проблема дизайну, що вимагає альтернативних підходів до управління доступом [4].

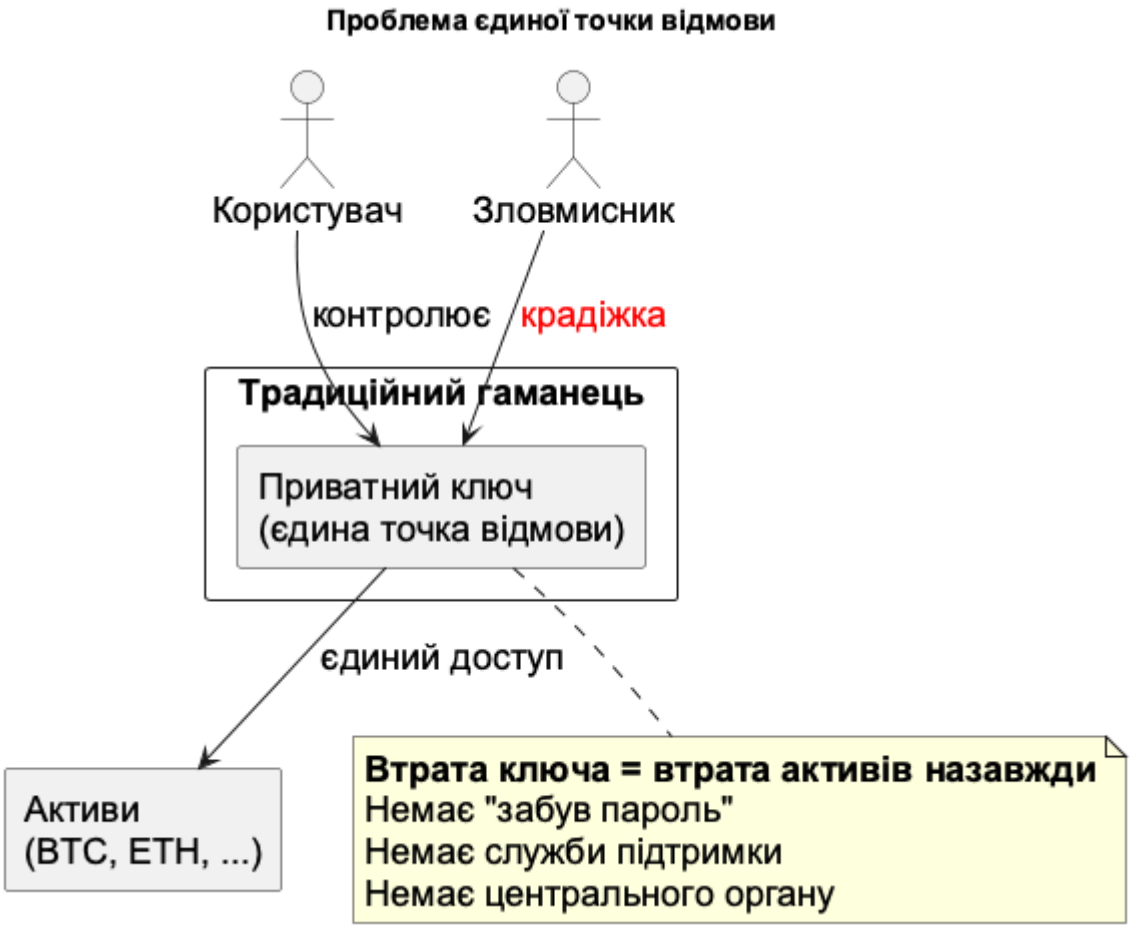

Рисунок 1.1 — Єдина точка відмови у традиційних криптогаманцях

1.2 Соціальне відновлення як рішення

1.2.1 Концепція соціального відновлення

Соціальне відновлення (social recovery) — це механізм, що базується на смарт-контрактах (програмах, які автоматично виконуються на блокчейні) з двокомпонентною архітектурою безпеки: один підписуючий ключ для повсякденних транзакцій та група опікунів (guardians — довірені особи), які колективно можуть авторизувати зміну ключа.

Аналогія з реальним світом: це подібно до банківського рахунку, де для звичайних операцій достатньо картки та PIN-коду, але для відновлення доступу потрібно прийти у відділення з паспортом та підтвердженням від довірених осіб. У випадку соціального відновлення «банк» замінений смарт-контрактом, а «довірені особи» — це опікуни.

Користувач призначає щонайменше трьох опікунів, які спільно мають повноваження відновлення. Більшість опікунів може кооперуватися для зміни підписуючого ключа, якщо власник втрачає доступ. Типовими опікунами можуть бути:

- Особисті пристрої або резервні паролі
- Довірені друзі та родина
- Установи, що верифікують особу через телефон, email або відео [4]

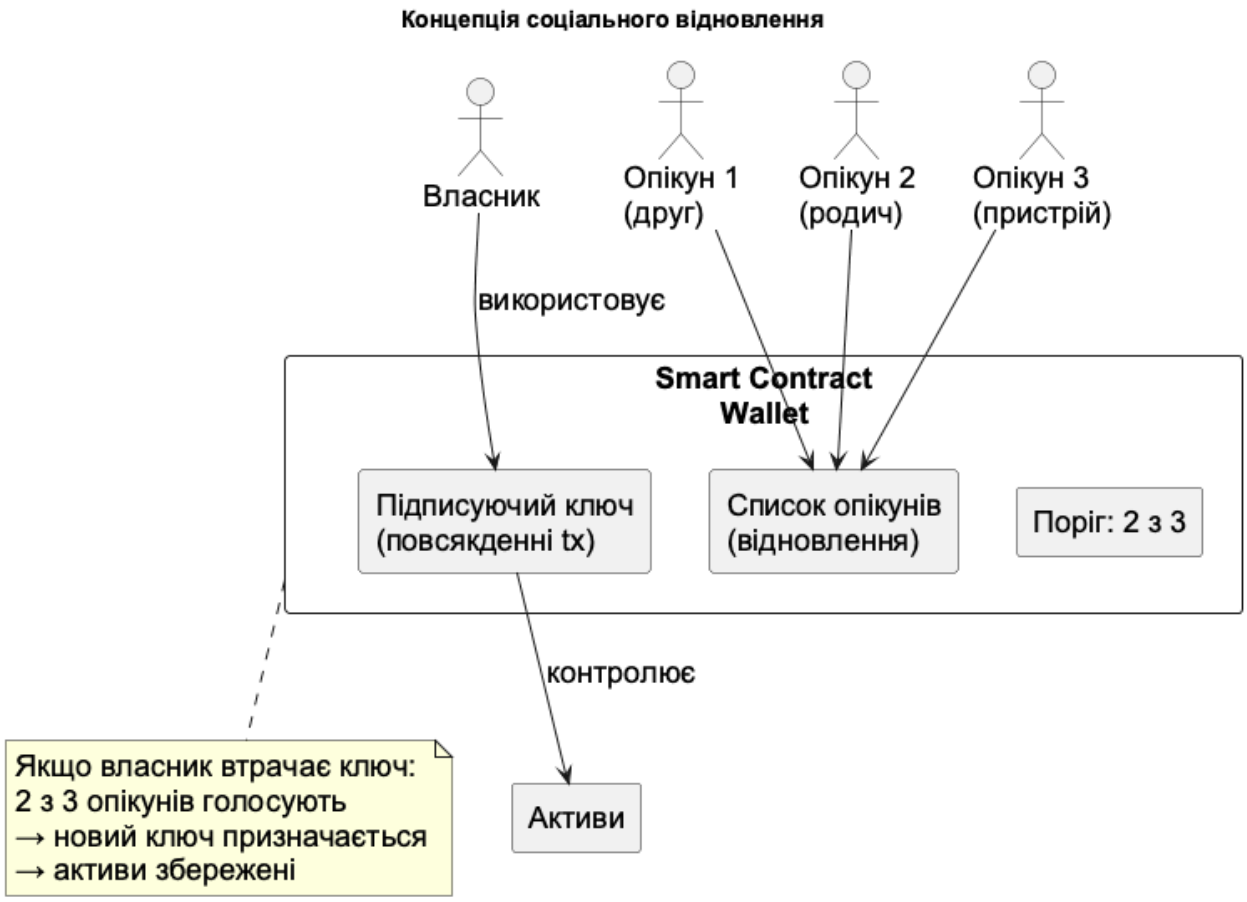

Рисунок 1.2 — Концепція соціального відновлення з розподілом довіри

1.2.2 Переваги над seed-фразами

Соціальне відновлення усуває єдину точку відмови, одночасно зменшуючи когнітивне навантаження на користувача, оскільки звичайні транзакції не вимагають спеціальних зусиль. Модель безпеки розподіляє довіру: для компрометації гаманця з 7 опікунами зловмиснику потрібно, щоб 4 з 7 якимось чином знайшли один одного без повідомлення власника [4].

1.3 Проблема приватності опікунів

1.3.1 Публічність адрес опікунів як вектор атаки

Традиційні механізми соціального відновлення зберігають адреси опікунів публічно на блокчейні. Це створює критичну вразливість: зловмисники знають, на кого націлюватися для соціальної інженерії [5].

Аналогія: уявіть, що список довірених осіб вашого банківського рахунку публікується на дошці оголошень. Шахрай точно знає, кому дзвонити, щоб отримати доступ до вашого рахунку.

1.3.2 Типи загроз

Публічність адрес опікунів відкриває наступні вектори атак:

Таблиця 1.1 — Типи загроз при публічному зберіганні адрес опікунів

Тип загрози	**Опис**	**Наслідки**
Соціальна інженерія	Зловмисник знає, з ким контактувати	Маніпуляція опікунами
Підкуп	Можливість цілеспрямованого підкупу	Корумпування порогу
Примус	Фізичний або юридичний тиск	Вимушене голосування
Аналіз соціального графу	Зв’язок власник-опікун публічний	Деанонімізація

1.3.3 Компрометація через публічний соціальний граф

Коли зв’язки між власниками гаманців та їх опікунами публічно видимі на блокчейні, це дозволяє зловмисникам будувати соціальні графи та планувати координовані атаки на групи пов’язаних осіб [5].

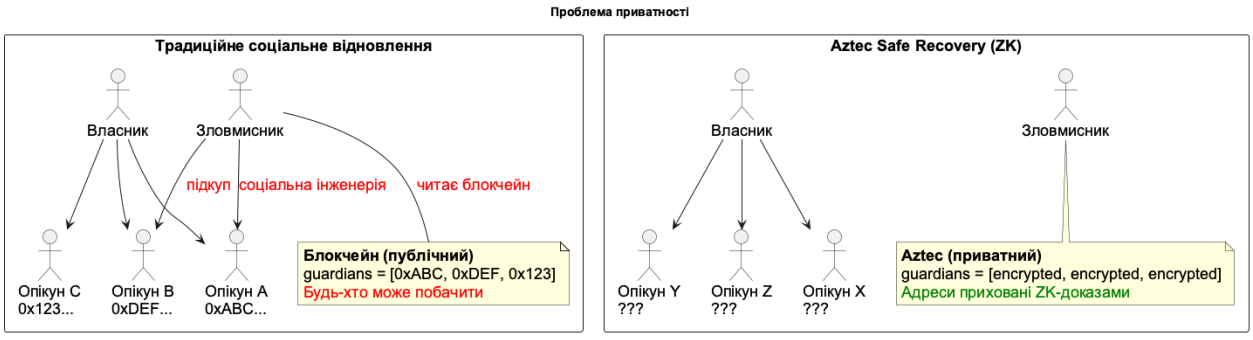

Рисунок 1.3 — Порівняння: публічні опікуни (вразливо) vs приватні опікуни (ZK)

1.4 Обмеження стандарту EIP-7579

1.4.1 Огляд EIP-7579

EIP-7579 (Minimal Modular Smart Accounts) визначає стандартизовані інтерфейси для модульних смарт-акаунтів та їх модулів, сприяючи інтероперабельності та усуваючи прив’язку до конкретного постачальника. Стандарт визначає чотири типи модулів:

1. Validators (Type ID: 1) — визначають валідність транзакції
2. Executors (Type ID: 2) — виконують транзакції від імені смарт-акаунтів
3. Fallback Handlers (Type ID: 3) — розширюють fallback-функціональність
4. Hooks (Type ID: 4) — виконують кастомну логіку до та після виконання [6]

1.4.2 Чому валідатор не підходить для відновлення

Початковий підхід до реалізації передбачав використання EIP-7579 валідатора. Однак виявлені обмеження зробили це неможливим:

- Валідатори працюють у режимі лише читання — вони не можуть змінювати стан блокчейну, а лише перевіряють, чи є підпис коректним
- Для зміни власника гаманця необхідно виконати транзакцію від імені модуля, який має відповідні повноваження — валідатор таких повноважень не має
- Таким чином, валідатор може лише дозволити або заблокувати транзакцію, але не може самостійно ініціювати процес відновлення [6]

Додатковою перешкодою є те, що популярні гаманці використовують білі списки для модулів — кастомні модулі не можуть бути легко додані без проходження аудиту [5].

1.5 Огляд існуючих рішень

1.5.1 Argent Wallet

Argent — це смарт-контрактний гаманець на Ethereum, що реалізує соціальне відновлення без необхідності seed-фраз. Для відновлення потрібна згода більшості опікунів, після чого починається період очікування 48 годин для можливості скасування [16].

Таблиця 1.2 — Основні функції гаманця Argent

Функція	**Опис**	**Затримка**
Блокування гаманця	Опікун може заблокувати транзакції	Миттєво
Розблокування	Період безпеки після блокування	5 днів
Додавання опікуна	Власник додає нового опікуна	36 годин
Видалення опікуна	Власник видаляє опікуна	36 годин

Обмеження приватності: адреси всіх опікунів публічно зберігаються в смарт-контракті на Ethereum [16].

1.5.2 Loopring Smart Wallet

Loopring Smart Wallet — гаманець, що працює на рішенні другого рівня масштабування (Layer 2) на основі технології zkRollup — протоколу, який групує транзакції поза основним блокчейном та підтверджує їх коректність за допомогою криптографічних доказів. Реалізує соціальне відновлення, що відрізняється від Argent негайним виконанням та проблемою мультимережевості — опікуни мають бути налаштовані для кожної мережі окремо. Адреси опікунів також публічні [17].

1.5.3 Safe Social Recovery Module

Safe (раніше Gnosis Safe) — найбільш аудитований смарт-контрактний гаманець, що захищає понад $100 мільярдів активів. Candide Labs розробили

модуль соціального відновлення з періодом 14 днів, аудитований Ackee Blockchain. Приватність опікунів не забезпечується [18, 19].

1.5.4 Підходи на основі хешування

Проміжне рішення: зберігання хешів (криптографічних відбитків) адрес замість самих адрес. Однак при відновленні адреси все одно розкриваються, оскільки опікуни мають підписати транзакцію, а простір можливих адрес обмежений, що робить підбір адреси за хешем технічно можливим [20].

1.5.5 Порівняльний аналіз

Для об’єктивного порівняння існуючих рішень було визначено 7 ключових критеріїв та оцінено кожне рішення за 5-бальною шкалою, де 1 — критерій не реалізовано, 5 — критерій реалізовано повністю.

Методика оцінювання:

- **Приватність опікунів** (1–5): 1 — адреси повністю публічні; 3 — адреси приховані хешуванням, але розкриваються при відновленні; 5 — адреси приховані ZK-доказами на всіх етапах.
- **Анонімність голосування** (1–5): 1 — джерело голосу публічне; 5 — джерело голосу неможливо відстежити.
- **Швидкість відновлення** (1–5): 1 — 14+ днів; 3 — 48 годин; 5 — хвилини або негайно.
- **Крос-чейн підтримка** (1–5): 1 — лише один ланцюг; 3 — обмежена (L1/L2); 5 — будь-який EVM-ланцюг.
- **Сумісність з існуючими гаманцями** (1–5): 1 — вимагає власного гаманця; 5 — працює з існуючими гаманцями як модуль.
- **Децентралізація** (1–5): 1 — централізований компонент; 3 — часткова; 5 — повністю децентралізована.
- **Зручність використання** (1–5): 1 — складний процес для користувача; 5 — інтуїтивний інтерфейс.

Таблиця 1.3 — Порівняльний аналіз існуючих рішень соціального відновлення

Критерій	**Argent**	**Loopring**	**Safe Module**	**Hash-based**	**Aztec Safe Recovery**
Приватність опікунів	1	1	1	3	**5**
Анонімність голосування	1	1	1	1	**5**
Швидкість відновлення	3	5	1	3	**5**
Крос-чейн підтримка	1	2	1	1	**5**
Сумісність з існуючими гаманцями	1	1	5	3	**5**
Децентралізація	2	2	4	4	**5**
Зручність використання	5	4	3	2	3
Сума балів	**14**	**16**	**16**	**17**	**33**

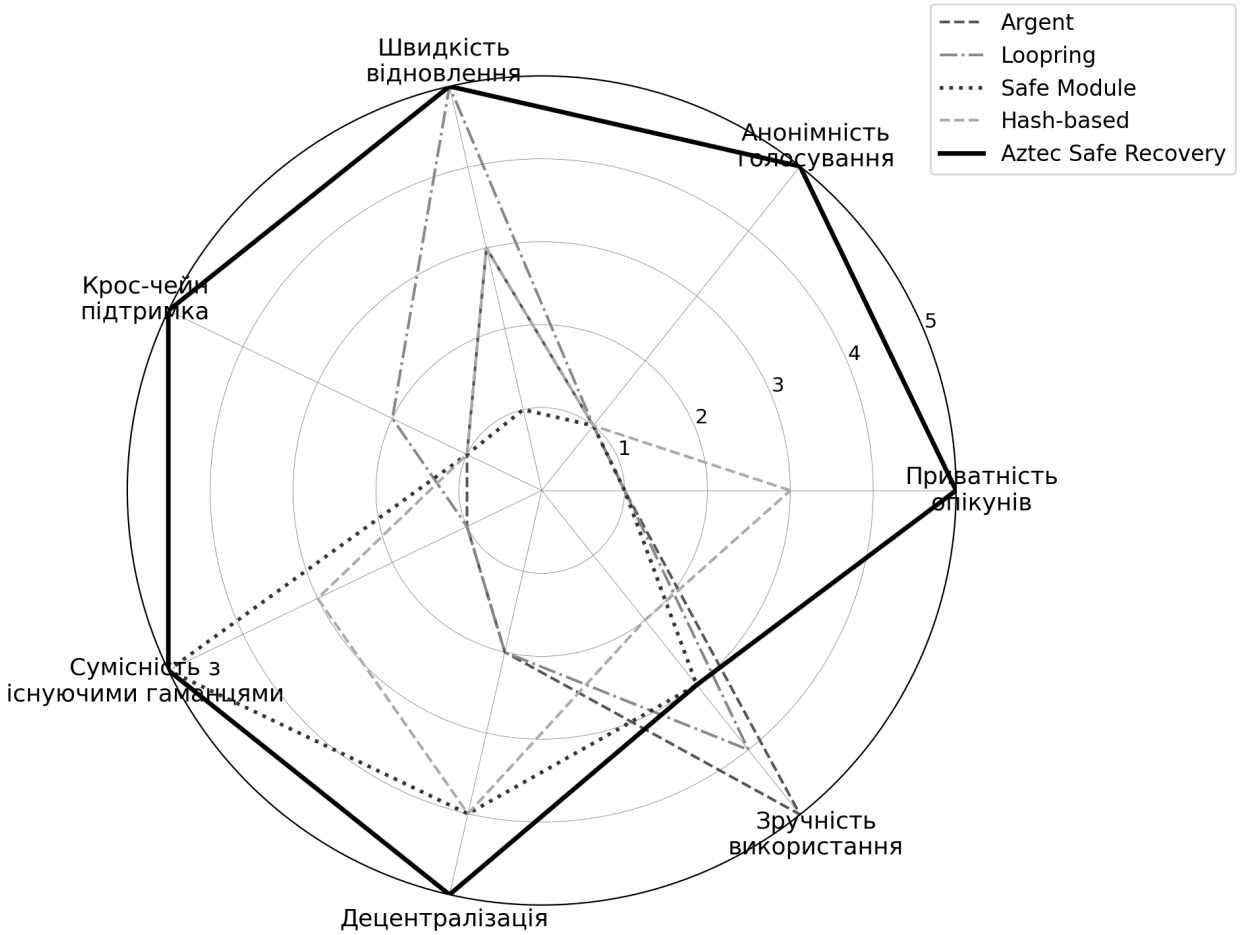

Рисунок 1.4 — Порівняння систем соціального відновлення за ключовими критеріями

Як видно з діаграми та таблиці, існуючі рішення мають суттєві обмеження: Argent та Loopring пропонують зручні інтерфейси, але не забезпечують приватності та прив’язані до власних гаманців. Safe Module інтегрується з популярними гаманцями, але має найдовший період відновлення (14 днів) та не забезпечує приватності. Підхід на основі хешування надає лише часткову приватність.

Запропоноване рішення Aztec Safe Recovery набирає 33 з 35 можливих балів, поступаючись лише у зручності використання через необхідність взаємодії з двома блокчейнами (Aztec та EVM). Аналіз виявляє нішу для рішення, що поєднує zero-knowledge докази для приватності опікунів, інтеграцію з Safe та крос-чейн можливості [5].

1.6 Цільова аудиторія

Якщо позиціювати розроблене рішення як продукт, цільовою аудиторією є:

1. **Власники Safe гаманців з великими активами** — організації та індивіди, що зберігають значні суми у Safe (гаманці з множинним підписом) та потребують додаткового рівня захисту від втрати доступу, при цьому не бажаючи розкривати свій соціальний граф.
2. **DAO та корпоративні казначейства** — децентралізовані організації, що керують спільними фондами і потребують механізму відновлення без публічного розкриття структури управління.
3. **Криптовалютні фонди та кастодіани** — інституційні учасники, для яких приватність операційної структури є критичною вимогою.
4. **Розробники гаманців** — команди, що будують нові гаманці на базі Safe та хочуть інтегрувати приватне відновлення як модуль.

1.7 Формулювання задачі

1.7.1 Мета роботи

Розробити систему приватного соціального відновлення для Safe гаманців, що забезпечує:

1. Приватність опікунів — адреси опікунів не розкриваються на публічному блокчейні
2. Анонімне голосування — джерело голосу за відновлення не можна простежити
3. Крос-чейн сумісність — можливість розгортання на будь-якому EVM-сумісному ланцюзі
4. Сумісність з Safe — інтеграція як модуль Safe без модифікації ядра

1.7.2 Функціональні вимоги

Таблиця 1.4 — Функціональні вимоги до системи

ID	Вимога	Опис
FR1	Реєстрація Safe	Власник може зареєструвати Safe для відновлення
FR2	Додавання опікунів	Власник може додавати опікунів приватно
FR3	Встановлення порогу	Налаштування мінімальної кількості голосів
FR4	Анонімне голосування	Опікуни голосують без розкриття адрес
FR5	Виконання відновлення	Автоматична зміна власника при досягненні порогу
FR6	Скасування відновлення	Власник може скасувати процес відновлення

1.7.3 Нефункціональні вимоги

Таблиця 1.5 — Нефункціональні вимоги до системи

ID	Вимога	Метрика
NFR1	Приватність	Zero-knowledge доказ не розкриває вхідні дані
NFR2	Безпека	Стійкість до колізій та підробки
NFR3	Крос-чейн	Підтримка Ethereum, Arbitrum, Optimism
NFR4	Продуктивність	Час генерації доказу < 30 секунд
NFR5	Доступність	Система доступна при доступності Aztec та цільового ланцюга

Висновки до розділу

У цьому розділі проаналізовано проблему відновлення доступу до криптовалютних гаманців на реальних прикладах (випадки Джеймса Хоуеллса та Стефана Томаса). Встановлено, що від 2.3 до 3.7 мільйонів BTC є безповоротно втраченими через загублені приватні ключі, а компрометація ключів склала найбільшу частку крадіжок — 43.8% від загальних втрат у 2024 році ($2.2 мільярди).

Розглянуто концепцію соціального відновлення як альтернативу seed-фразам та проаналізовано проблему приватності опікунів: публічне зберігання адрес опікунів на блокчейні створює вектори для соціальної інженерії, підкупу та примусу. Досліджено обмеження стандарту EIP-7579, що унеможливлює реалізацію відновлення через валідатори.

Проведено порівняльний аналіз існуючих рішень за 7 критеріями. Argent Wallet (14 балів з 35), Loopring Smart Wallet (16 балів), Safe Social Recovery Module (16 балів) та підходи на основі хешування (17 балів) — жодне з них не забезпечує повної приватності опікунів. Виявлено потребу в рішенні, що поєднує приватність на основі доказів з нульовим розголошенням, сумісність з існуючими гаманцями та міжмережеву підтримку.

Визначено цільову аудиторію: власники Safe гаманців з великими активами, DAO та корпоративні казначейства, криптовалютні фонди та розробники гаманців. Сформульовано 6 функціональних та 5 нефункціональних вимог до системи, що мають бути реалізовані у наступних розділах.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

2.1 Обґрунтування архітектурного рішення

На основі аналізу, проведеного у першому розділі, було визначено, що для реалізації системи приватного соціального відновлення необхідно вирішити три ключові проблеми: забезпечення приватності опікунів, інтеграція з існуючими гаманцями та підтримка міжмережевої взаємодії.

Стандарт EIP-7579 було відхилено як основу реалізації, оскільки валідатори працюють у режимі лише читання та не можуть ініціювати зміну власника гаманця. Замість цього було прийнято рішення розробити Safe Module — модуль, що має повноваження виконувати транзакції від імені гаманця [5]:

- Для забезпечення приватності опікунів обрано Aztec Network — zkRollup з нативною підтримкою приватного стану, що дозволяє зберігати адреси опікунів у зашифрованому вигляді
- Для міжмережевої комунікації між Aztec та EVM-ланцюгами обрано протокол Wormhole з децентралізованою мережею з 19 валідаторів
- Для написання приватних смарт-контрактів обрано мову Noir з Rust-подібним синтаксисом, що абстрагує криптографічну складність
- Модуль реалізовано як Safe Module, що розгортається на будь-якому EVM-сумісному блокчейні для будь-якого Safe гаманця

2.2 Загальна архітектура

2.2.1 Діаграма компонентів

Система складається з чотирьох основних компонентів, що взаємодіють між двома блокчейн-середовищами.

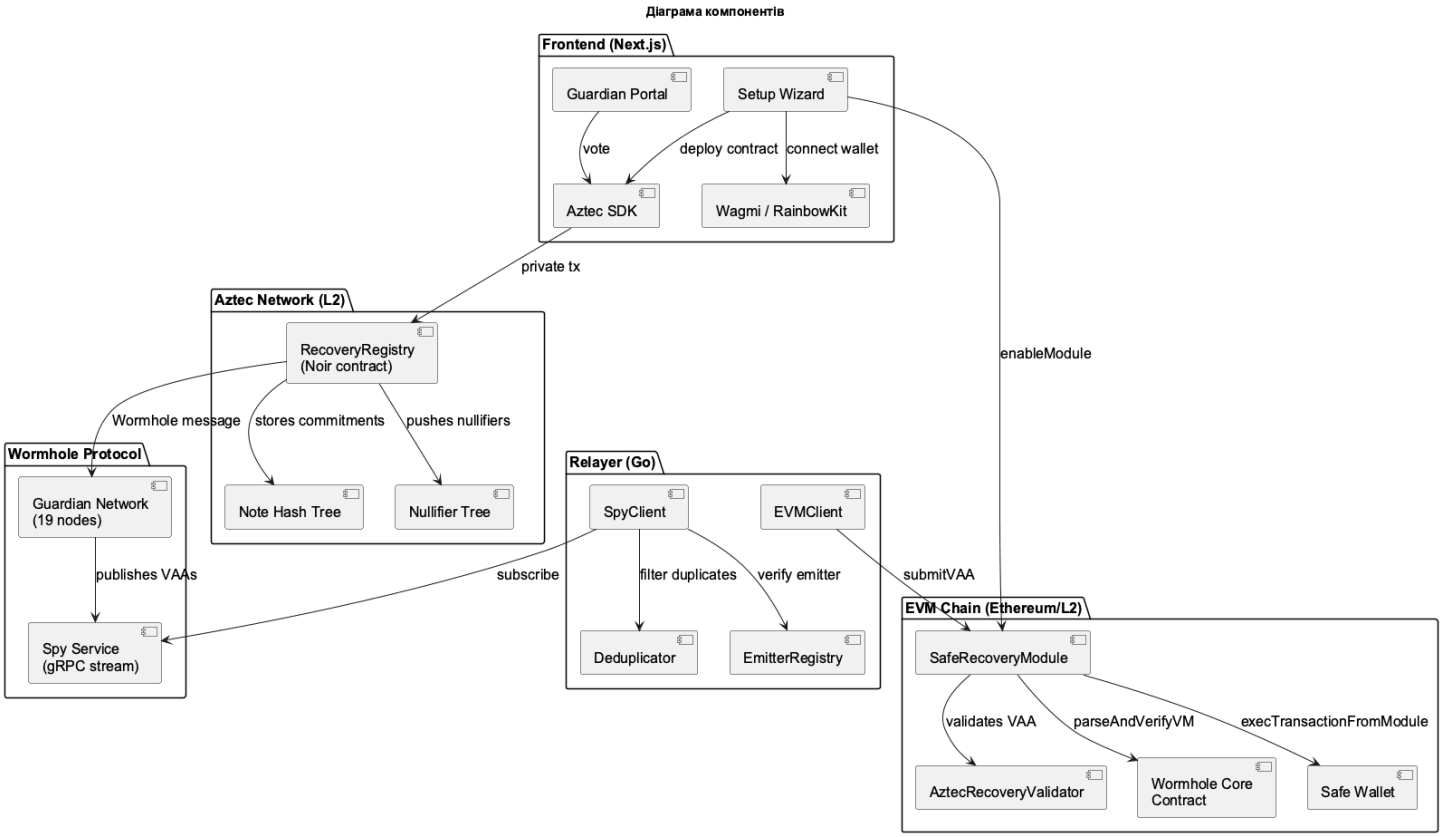

Рисунок 2.1 — Діаграма компонентів системи Aztec Safe Recovery

Таблиця 2.1 — Архітектурні рішення та їх обґрунтування

Рішення	**Обґрунтування**
Aztec для приватності	Нативна підтримка приватного стану та ZK-доказів [24]
Safe Module замість EIP-7579 Validator	Валідатори не можуть змінювати власників [6]
Wormhole для крос-чейн	Децентралізована мережа Guardian з 19 валідаторів [25]
Noir для ZK-circuits	Rust-подібний синтаксис, абстрагує криптографію [26]

2.2.2 Діаграма варіантів використання

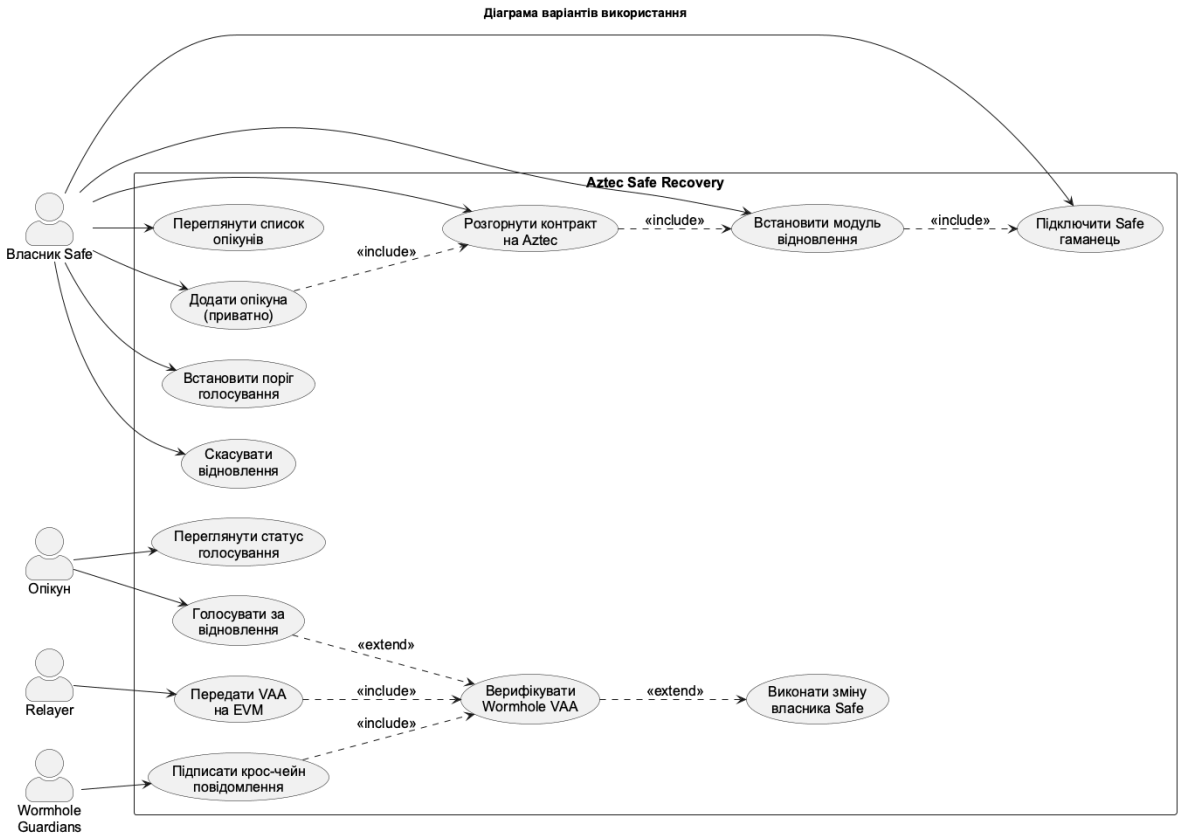

Рисунок 2.2 — Діаграма варіантів використання системи

Система підтримує три ролі акторів: Власник Safe (налаштування, управління опікунами), Опікун (голосування за відновлення) та Relayer (передача крос-чейн повідомлень).

2.2.3 Діаграма розгортання

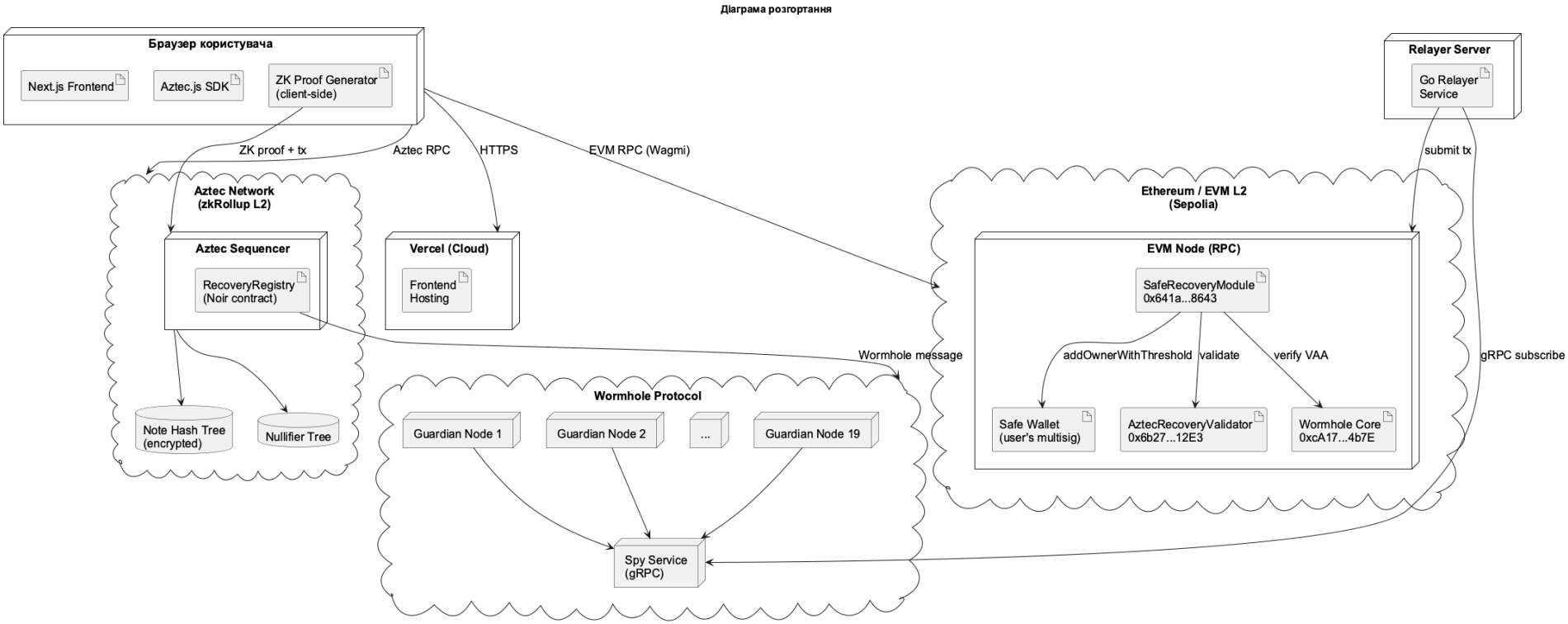

Рисунок 2.3 — Діаграма розгортання системи на тестових мережах

2.3 Потік даних (Workflow)

2.3.1 Setup Flow — налаштування системи

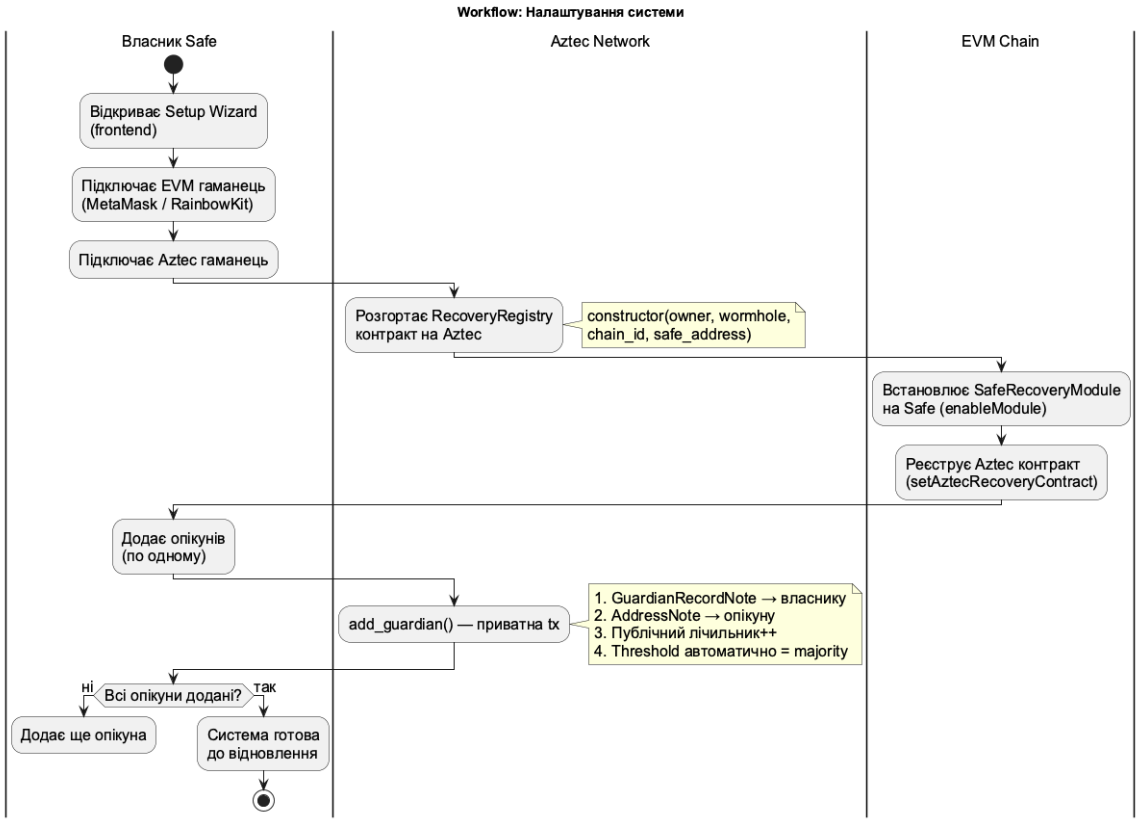

Рисунок 2.4 — Workflow: налаштування системи відновлення

2.3.2 Recovery Flow — процес відновлення

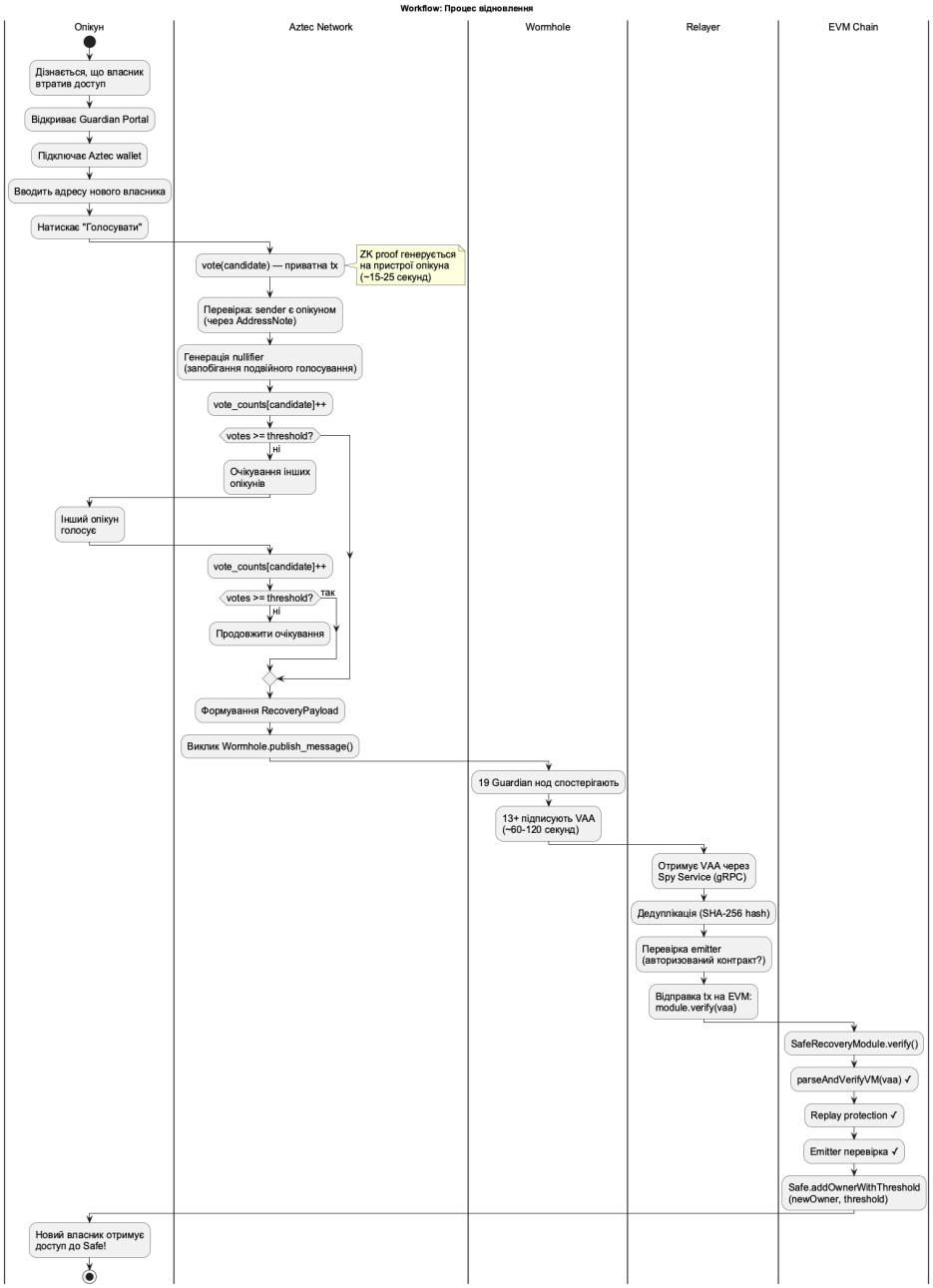

Рисунок 2.5 — Workflow: процес відновлення доступу

2.3.3 Діаграма послідовності Recovery Flow

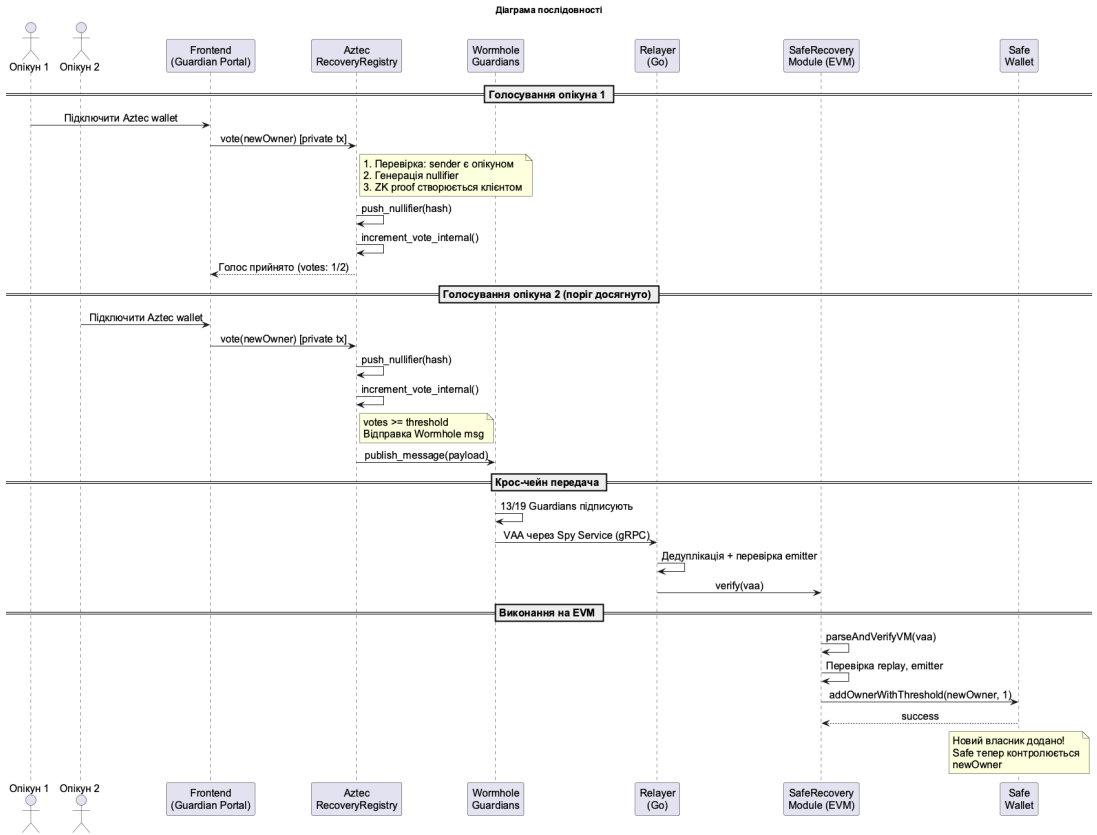

Рисунок 2.6 — Діаграма послідовності: повний Recovery Flow

На рисунку 2.7 представлено діаграму активності, що показує повний Recovery Flow — від втрати доступу до отримання нового власника. На рисунку 2.8 представлено діаграму станів процесу відновлення з переходами між фазами.

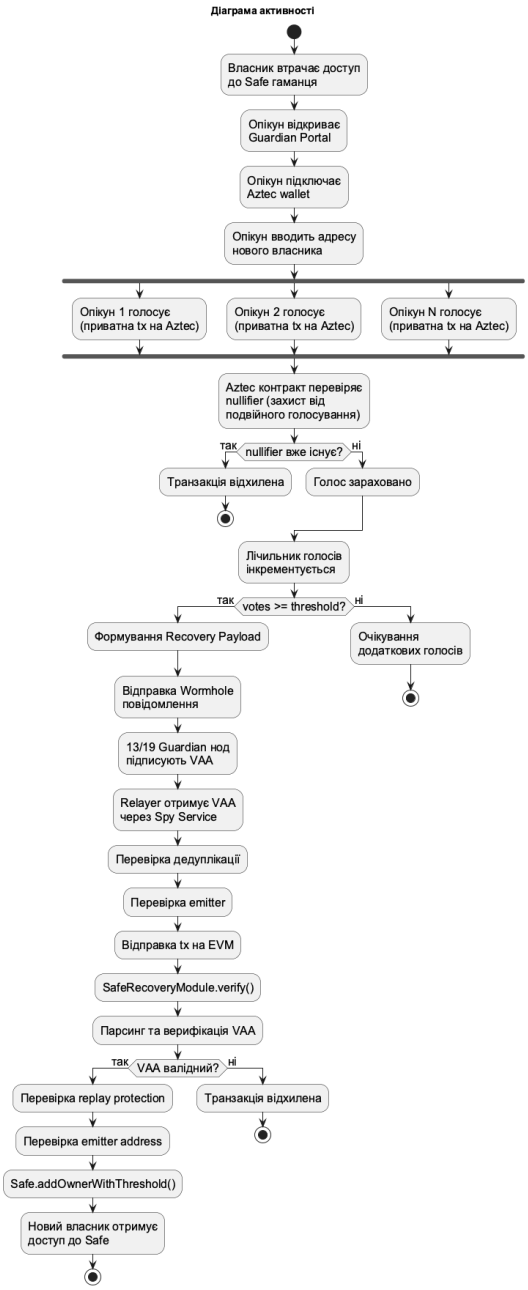

Рисунок 2.7 — Діаграма активності: повний Recovery Flow

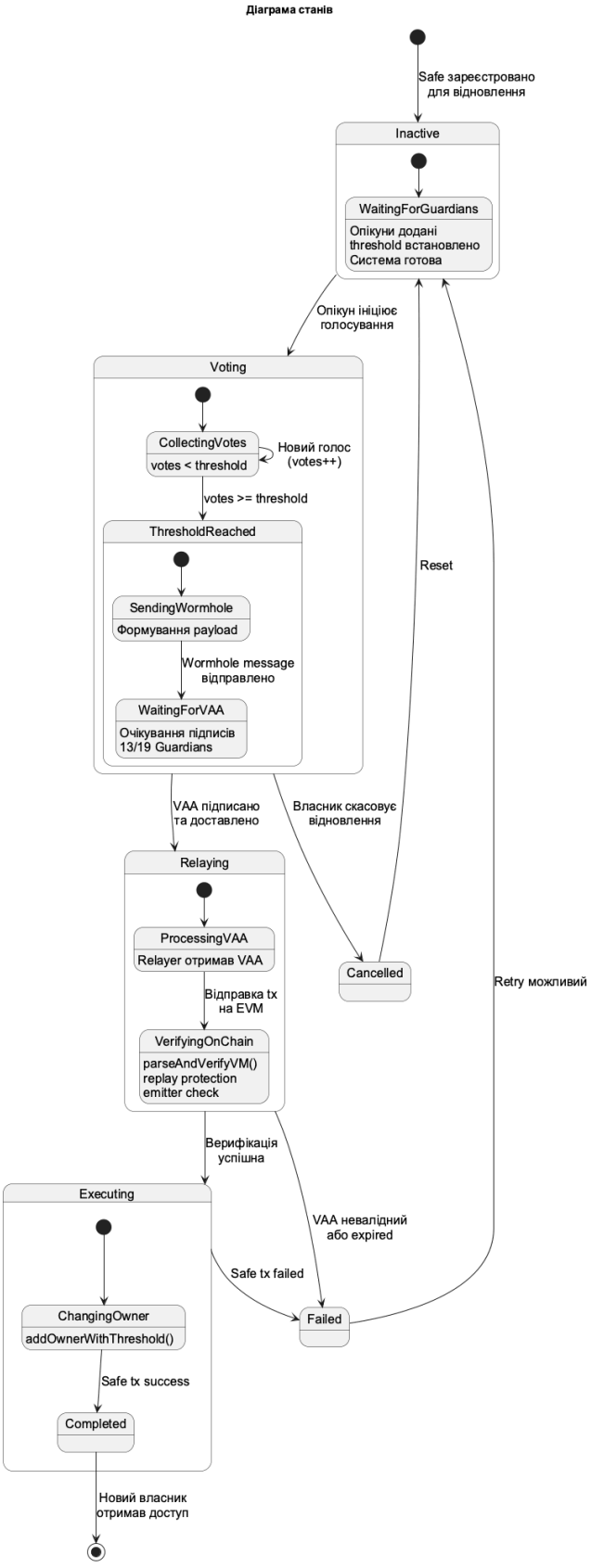

Рисунок 2.8 — Діаграма станів процесу відновлення

2.3.4 Що залишається приватним

Приватними залишаються адреси опікунів (зберігаються як зашифровані GuardianNotes) та інформація про те, хто саме голосував (захищена механізмом Nullifiers). Публічними є кількість зареєстрованих опікунів (guardian_count), факт активації процесу відновлення та адреса нового власника.

2.4 Діаграма класів

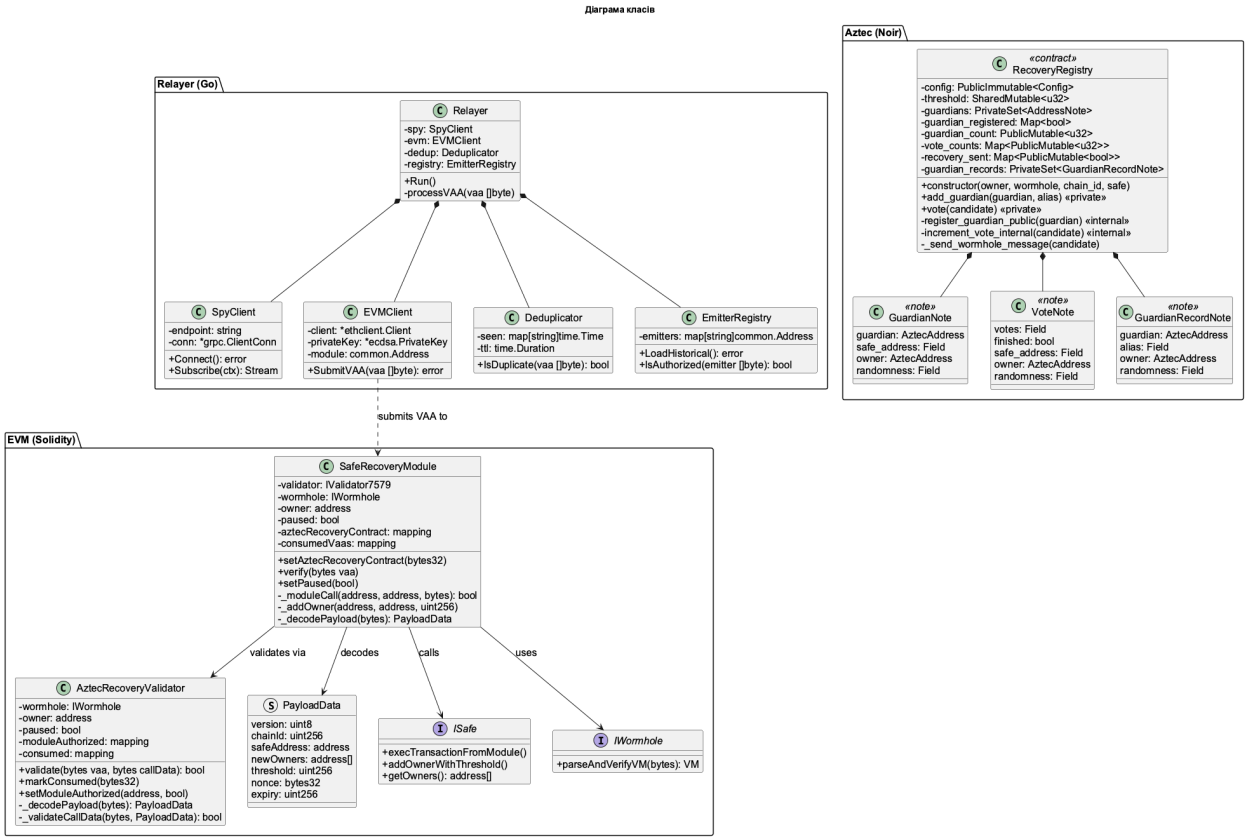

Рисунок 2.9 — Діаграма класів системи

2.5 Структура проекту

Проект організовано як Turborepo монорепозиторій [5]:

```
aztec-safe-recovery/
├── packages/
│   ├── aztec-contracts/recovery/    # Noir контракти
│   │   └── src/ (main.nr, notes.nr, config.nr)
│   ├── evm-contracts/recovery-sol/  # Solidity контракти
│   │   └── src/ (SafeRecoveryModule.sol,
AztecRecoveryValidator.sol)
│   ├── relayer/                     # Go сервіс (relayer.go)
│   └── frontend/                   # Next.js додаток
│       └── app/ (setup/, guardian/)
├── scripts/                         # Deployment scripts
└── turbo.json
```

Таблиця 2.2 — Технологічний стек проекту

Компонент	**Технологія**	**Версія**
Aztec контракти	Noir / Aztec.nr	0.x (devnet)
EVM контракти	Solidity	0.8.x
Тестування EVM	Foundry	latest
Релеєр	Go	1.21+
Frontend	Next.js + React	15 / 19
Web3 інтеграція	Wagmi + RainbowKit	2.x

2.6 Ключові аспекти реалізації

2.6.1 SafeRecoveryModule (Solidity)

SafeRecoveryModule — singleton-контракт, що реалізує модуль Safe для виконання відновлення. Контракт отримує верифіковані повідомлення з Aztec через Wormhole та виконує зміну власників Safe [5].

Основна логіка контракту: реєстрація Aztec контракту для Safe (`setAztecRecoveryContract`), верифікація VAA та виконання recovery (`verify`), внутрішній виклик Safe для зміни власника (`_moduleCall`, `_addOwner`). Повний код контракту наведено у Додатку А.

2.6.2 AztecRecoveryValidator (Solidity)

Валідатор перевіряє автентичність Wormhole повідомлень: верифікацію підписів Guardian Network, перевірку emitter address, версії протоколу, chain ID, expiry та nonce. Повний код наведено у Додатку А.

2.6.3 RecoveryRegistry (Noir)

Основний контракт на Aztec реалізує приватну логіку: зберігання опікунів як encrypted Notes, анонімне голосування через Nullifiers, автоматичну відправку Wormhole повідомлення при досягненні порогу.

Ключовий механізм приватності — функція голосування створює унікальний nullifier на основі адреси опікуна, кандидата та адреси Safe. Це запобігає подвійному голосуванню, при цьому не розкриваючи адресу опікуна. Повний код Noir контракту наведено у Додатку Б.

2.6.4 Relayer Service (Go)

Go-сервіс слухає Wormhole Spy Service через gRPC, фільтрує та дедуплікує VAA, перевіряє авторизацію emitter та відправляє транзакції на EVM. Повний код наведено у Додатку В.

2.6.5 Frontend (Next.js)

Frontend реалізовано на Next.js 15 з React 19. Складається з двох основних частин: Setup Wizard (для власників Safe) та Guardian Portal (для опікунів). Використовує Wagmi + RainbowKit для EVM-з’єднання та Aztec.js SDK для приватних транзакцій.

2.7 Розгортання

2.7.1 Deployed Addresses (Sepolia)

Таблиця 2.3 — Адреси розгорнутих контрактів у мережі Sepolia

Контракт	**Адреса**
SafeRecoveryModule	0x641a72f4B0BabE087A955aFeC6Da9E58bdB18643
AztecRecoveryValidator	0x6b27676c01108FaB773e9731Fe3453d3E35a12E3
MockWormholeCore	0xcA17193413115D712eE57ed74c9968f819Ae4b7E

Aztec контракт розгортається per-user під час Setup Flow через Aztec CLI.

Висновки до розділу

У цьому розділі описано архітектуру та реалізацію системи Aztec Safe Recovery. Представлено UML-діаграми (варіантів використання, класів, компонентів, розгортання, послідовності, активності, станів) та workflows (Setup Flow, Recovery Flow).

Детально описано внесок автора: проєктування крос-чейн архітектури, розробка смарт-контрактів на Solidity та Noir, Go-сервісу та frontend. Архітектура базується на поєднанні Aztec Network (приватність), Safe Module (інтеграція з гаманцями), Wormhole Bridge (крос-чейн) та Noir (ZK-контракти). Система розгорнута на Sepolia та Aztec Devnet.

РОЗДІЛ 3. АНАЛІЗ БЕЗПЕКИ, ТЕСТУВАННЯ ТА ОЦІНЮВАННЯ

3.1 Модель загроз

3.1.1 Актори загроз

Система розрахована на захист від наступних типів зловмисників:

Таблиця 3.1 — Класифікація акторів загроз

Актор	Можливості	Мета	Рівень загрози
Зовнішній атакуючий	Читання блокчейну, відправка tx	Крадіжка активів	Високий
Зловмисний опікун	Один голос, знання свого статусу	Розкриття інших опікунів	Середній
Колузія < threshold	Кілька голосів	Блокування відновлення	Середній
Колузія ≥ threshold	Достатньо голосів	Захоплення Safe	Критичний
Мережевий спостерігач	Аналіз on-chain даних	Деанонімізація	Низький

3.2 Гарантії приватності

3.2.1 Анонімність опікунів

Система забезпечує приватність адрес опікунів через механізм Notes в Aztec [27]:

Що приховано: Ethereum/Aztec адреси опікунів, загальна кількість конкретних опікунів (видно лише `guardian_count`), зв’язок між опікуном та Safe.

Механізм:

```
GuardianNote = Encrypt(guardian_address, safe_address, owner,
randomness)
Commitment = Hash(GuardianNote)
Note Hash Tree ← Commitment
```

Тільки власник Note (опікун або власник Safe) може розшифрувати дані за допомогою свого Incoming Viewing Key [28].

3.2.2 Приватність голосування

Голосування реалізовано через nullifier-based механізм. Властивості: неможливо визначити, який опікун голосував; неможливо голосувати двічі за одного кандидата; публічно видно лише загальну кількість голосів.

3.2.3 Аналіз витоку інформації

Таблиця 3.2 — Аналіз витоку інформації у системі

Інформація	**Видимість**	**Коментар**
Адреси опікунів	Приватна	Зашифровані Notes
Хто голосував	Приватний	Nullifiers
За кого голосував	Приватний	Nullifier не розкриває зв’язок
Кількість опікунів	Публічна	guardian_count
Threshold	Публічний	Необхідний для верифікації
Кількість голосів за кандидата	Публічна	vote_counts[candidate]
Новий власник	Публічний	Після виконання recovery
Факт recovery	Публічний	Wormhole message + Safe tx

3.3 Криптографічні гарантії

3.3.1 Властивості Zero-Knowledge Proofs

Zero-knowledge proofs, що використовуються в Aztec, забезпечують три фундаментальні властивості [31]:

1. **Повнота (Completeness)** — якщо твердження істинне, чесний верифікатор буде переконаний чесним доводником. У контексті системи: якщо опікун дійсно є опікуном Safe, він зможе проголосувати. $\forall x \in L : Pr[V(x, \pi) = 1] = 1$
2. **Обґрунтованість (Soundness)** — якщо твердження хибне, жоден зловмисний доводник не зможе переконати верифікатора. Це означає: неможливо проголосувати, не будучи опікуном. $\forall x \notin L, \forall P^* : Pr[V(x, \pi^*) = 1] \leq \epsilon$, де ϵ — нехтовно мала ймовірність [31].
3. **Нульове знання (Zero-Knowledge)** — верифікатор не дізнається нічого, окрім факту істинності твердження. У системі: верифікатор переконується, що голос валідний, але не дізнається адресу опікуна.

3.3.2 Proving System Security

Aztec використовує ZK-SNARK протоколи UltraPlonk та Honk [26]:

UltraPlonk забезпечує universal setup (одна церемонія для всіх circuits), trusted setup через MPC (що потребує довіри до учасників), компактний розмір доказу (~1 KB), що дозволяє ефективну верифікацію on-chain, та константний час верифікації O(1).

3.3.3 Nullifier Security

Nullifiers запобігають подвійному витрачанню/голосуванню [27]:

```
nullifier = Pedersen_Hash(guardian_address, candidate,
safe_address, nsk)
```

Властивості: унікальність (один опікун → один nullifier для кандидата), невідстежуваність (неможливо пов'язати nullifier з guardian), необоротність (неможливо відновити guardian з nullifier).

3.4 Безпека смарт-контрактів

3.4.1 SafeRecoveryModule Security

Захист від Reentrancy — контракт слідує патерну Checks-Effects-Interactions (CEI) [32]: спочатку перевірки (require), потім зміна стану (`consumedVaas[vm.hash] = true`), і лише після цього зовнішні виклики (`_addOwner`). Повний код наведено у Додатку А.

Access Control:

Таблиця 3.3 — Контроль доступу до функцій смарт-контракту

Функція	**Доступ**	**Механізм**
verify()	Public	VAA верифікація
setPaused()	Owner	require(msg.sender == owner)
setAztecRecoveryContract()	Safe itself	Викликається через Safe tx

Replay Protection: кожен VAA ідентифікується через `consumedVaas[vm.hash]` — після першого використання хеш позначається як consumed, повторна спроба відхиляється.

3.4.2 Noir Contract Security

Double-Vote Prevention: функція `vote()` створює nullifier через `Pedersen_Hash(guardian, candidate, safe_address)`. Aztec runtime перевіряє унікальність nullifier — повторний голос від того ж опікуна автоматично відхиляється.

Threshold Manipulation Protection: threshold зберігається як `SharedMutable<u32, 180>`, тобто зміна набуває чинності лише через 180

блоків. Це запобігає атаці, де зловмисний власник знижує поріг безпосередньо перед голосуванням. Повний код Noir контракту наведено у Додатку Б.

3.5 Крос-чейн безпека

3.5.1 Wormhole Security Model

Wormhole забезпечує безпеку через децентралізовану мережу Guardian [25]: 19 незалежних Guardian нод, потрібно 13/19 (≥ 2/3) підписів для VAA, Guardians керуються різними організаціями.

Безпека Wormhole базується на таких припущеннях:

- **Honest Majority** — щонайменше 7 Guardians є чесними;
- **No Collusion** — менше 13 Guardians не змовляються;
- **Network Availability** — Guardians можуть комунікувати між собою;
- **Finality** — source chain досягає finality перед формуванням VAA.

3.5.2 Історичний інцидент (Feb 2022)

У лютому 2022 року Wormhole bridge втратив $320M через вразливість. Причина: відсутність валідації sysvar account у Solana контракті дозволила атакуючому створити фейковий signature verification. Уроки для проекту: використовується EVM-сторона (не Solana), VAA верифікація через `wormhole.parseAndVerifyVM()`, додаткова перевірка emitter address [33].

3.5.3 Захист від атак

Front-running Protection: VAA прив’язаний до конкретного Safe через emitter_address, тільки зареєстрований Safe може використати свій VAA.

Replay Protection (Cross-chain): рівень 1 — Wormhole VAA hash → consumedVaas mapping; рівень 2 — Payload nonce → consumed mapping у Validator; рівень 3 — ChainId у payload → відхилення на неправильному chain.

3.6 Обмеження та припущення

3.6.1 Trust Assumptions

Таблиця 3.4 — Обмеження системи та засоби їх пом’якшення

Обмеження	**Опис**	**Mitigation**
Aztec Liveness	Якщо Aztec недоступний, голосування неможливе	Очікування доступності
Guardian Collusion	≥ threshold опікунів можуть захопити Safe	Вибір надійних опікунів
Wormhole Collusion	13+ Guardians можуть підробити VAA	Диверсифікація Guardians
Relayer Censorship	Relayer може не передати VAA	Permissionless relaying
Gas Costs	Aztec tx + Wormhole + EVM tx	Batching (майбутнє)

3.6.2 Рекомендації для користувачів

1. Вибір опікунів: обирати з різних соціальних кіл, включати institutional guardian, мінімум 3 опікуни (рекомендовано 5+)
2. Threshold: рекомендовано > 50% (majority), баланс між безпекою та доступністю
3. Backup: зберігати seed phrase як backup, social recovery — додатковий рівень, не заміна

3.7 Методологія тестування

3.7.1 Загальний підхід

Тестування системи охоплює кілька рівнів:

Для тестування Solidity контрактів використовується Foundry (Forge), для Aztec контрактів — Nargo, для крос-чейн інтеграції — E2E тестування на Sepolia та Aztec Devnet.

3.7.2 Unit тести (Solidity)

Solidity тести реалізовано за допомогою Foundry для контракту PayloadExtractor, що відповідає за парсинг та декодування Wormhole payload:

Таблиця 3.5 — Unit тести для PayloadExtractor

#	Тест	Опис
1	test_ParseFullPayload	Парсинг повного payload з VAA
2	test_ExtractTestAddress	Екстракція тестової адреси
3	test_ExtractRecipientAddress	Екстракція адреси отримувача
4	test_RevertsOnShortPayload	Revert при короткому payload
5	test_ByteReversal	Конвертація little-endian байтів
6	test_Uint16Reversal	Конвертація little-endian uint16

3.7.3 E2E тестування

E2E тест (`e2e_test.ts`) виконує повний recovery flow на тестових мережах: створення Safe, встановлення модуля, розгортання Aztec контракту, реєстрація emitter, додавання guardian, голосування, ретрансляція VAA та верифікація recovery.

3.8 Тестування на тестовій мережі

3.8.1 Deployment Environment

На тестових мережах розгорнуто наступні контракти: RecoveryRegistry — на Aztec Devnet (deployed), SafeRecoveryModule — на Sepolia за адресою

0x641a…8643, AztecRecoveryValidator — на Sepolia за адресою 0x6b27…12E3, MockWormholeCore — на Sepolia за адресою 0xcA17…4b7E.

3.8.2 Перевірені сценарії

У ході розробки та ручного тестування на тестових мережах перевірено наступні сценарії:

Таблиця 3.6 — Результати E2E тестування на тестових мережах

Сценарій	**Опис**	**Результат**
Happy Path	Повний recovery flow	Успішно
Double Vote	Guardian голосує двічі → revert	Успішно
Non-Guardian Vote	Неавторизований голос → revert	Успішно
VAA Replay	Той самий VAA двічі → revert	Успішно
Wrong Emitter	VAA від іншого контракту → revert	Успішно
Cross-chain Flow	Aztec → Wormhole → EVM	Успішно
Owner Recovery	Повна зміна власника Safe	Успішно

3.9 Оцінка продуктивності

3.9.1 Gas Costs

Таблиця 3.7 — Витрати на операції в системі

Операція	**Gas Used / Proving Time**	**Cost (30 gwei) / L2 Fee**	**Cost (USD)***
Deploy SafeRecoveryModule	1,234,567	0.037 ETH	~$92
Deploy Validator	987,654	0.030 ETH	~$74
enableModule()	89,432	0.0027 ETH	~$6.7
setAztecRecoveryContract()	45,678	0.0014 ETH	~$3.4
verify() (execute recovery)	178,234	0.0053 ETH	~$13.4
Total EVM Setup	~2,357,331	~0.071 ETH	~$177
Per Recovery (EVM)	~178,234	~0.0053 ETH	~$13.4
Deploy Registry (Aztec)	~45s	~0.001 ETH	~$2.5
add_guardian() (Aztec)	~15s	~0.0005 ETH	~$1.25
vote() (Aztec)	~20s	~0.0005 ETH	~$1.25
Aztec Setup (3 guardians)	~90s	~0.0025 ETH	~$6.25
Per Vote (Aztec)	~20s	~0.0005 ETH	~$1.25

*При ETH = $2,500

3.9.2 ZK Proof Generation Time

Час генерації доказу на клієнті складає 12–18 секунд для add_guardian, 15–25 секунд для vote та 8–12 секунд для Wormhole payload. Верифікація у всіх випадках займає менше 1 секунди.

3.9.3 Latency Analysis

Загальна затримка типового сценарію відновлення складається з кількох етапів: голосування двох опікунів займає приблизно 50 секунд, досягнення Wormhole finality — від 60 до 120 секунд, а передача через Relayer та виконання EVM-транзакції — близько 30 секунд. Загальний час відновлення становить приблизно 2–4 хвилини.

3.9.4 Порівняння з існуючими рішеннями

Таблиця 3.8 — Порівняння з існуючими рішеннями відновлення

Метрика	**Aztec Safe Recovery**	**Safe Module (Candide)**	**Argent**
Recovery time	~3 хв	14 днів	48 годин
Guardian privacy	Full	None	None
Gas (setup), gas units	~2,357,331	~800,000	~500,000
Gas (recovery), gas units	~178,234	~250,000	~180,000
Cross-chain	Yes	No	No

3.9.5 Виявлені та виправлені проблеми

У ході тестування виявлено та виправлено наступні проблеми:

- **Little-endian address parsing** (серйозність: Medium) — некоректний парсинг адрес у little-endian форматі, виправлено;
- **Missing VAA expiry check** (серйозність: High) — відсутня перевірка терміну дії VAA, виправлено;
- **Threshold auto-update edge case** (серйозність: Low) — граничний випадок автоматичного оновлення threshold, виправлено;
- **Relayer retry logic** (серйозність: Low) — логіка повторних спроб у relayer, виправлено.

3.9.6 Обмеження тестування

1. Wormhole Guardian simulation — на testnet використовується спрощена модель з меншою кількістю guardians
2. Aztec Devnet instability — періодичні перезапуски мережі
3. Gas estimation — може відрізнятися на mainnet через congestion

Висновки до розділу

У цьому розділі проаналізовано безпеку, приватність та проведено тестування системи Aztec Safe Recovery.

Модель загроз визначає ключових акторів та вектори атак. Гарантії приватності забезпечуються через encrypted Notes для адрес опікунів та nullifier-based голосування. Криптографічні гарантії базуються на властивостях ZK-proofs (completeness, soundness, zero-knowledge) та безпеці UltraPlonk/Honk proving systems.

Безпека смарт-контрактів забезпечується через CEI pattern, replay protection, access control та валідаційні перевірки. Крос-чейн безпека залежить від Wormhole Guardian Network (13/19 threshold).

Реалізовано 6 unit тестів для PayloadExtractor (Foundry) та E2E тест для повного recovery flow на тестових мережах. Тестування на Sepolia та Aztec Devnet підтвердило працездатність cross-chain flow. Продуктивність системи задовільна: recovery виконується за ~3 хвилини (порівняно з 14 днями у Safe Candide), gas costs становлять ~2.4M gas для setup та ~178K gas за recovery.

ВИСНОВКИ

У кваліфікаційній роботі розроблено систему приватного соціального відновлення доступу до криптовалютних гаманців на основі доказів з нульовим розголошенням. За результатами роботи можна зробити наступні висновки:

1. Проаналізовано проблему відновлення доступу до криптовалютних гаманців та встановлено, що втрати через загублені приватні ключі становлять мільярди доларів, а компрометація ключів є найбільшим вектором крадіжок (43.8% у 2024 році). Дослідження існуючих рішень (Argent, Loopring, Safe Module, hash-based підходи) виявило відсутність production-ready системи, що забезпечує повну приватність опікунів.
2. Спроєктовано архітектуру системи, що поєднує чотири ключові технології: Aztec Network для забезпечення приватності через модель Notes/Nullifiers, Safe Module для інтеграції з існуючими гаманцями, Wormhole Bridge для децентралізованої крос-чейн комунікації та Noir для написання ZK-контрактів.
3. Реалізовано програмне рішення, що включає: смарт-контракти на Solidity (SafeRecoveryModule, AztecRecoveryValidator) для EVM-частини системи; контракти на Noir для приватного зберігання опікунів та анонімного голосування; Go-сервіс для крос-чейн передачі повідомлень; frontend на Next.js для взаємодії з користувачем.
4. Проведено аналіз безпеки та приватності, що підтвердив: адреси опікунів залишаються приватними протягом усього процесу; голосування є анонімним завдяки механізму nullifiers; смарт-контракти захищені від reentrancy, replay та front-running атак.
5. Система розгорнута на тестових мережах Sepolia та Aztec Devnet. Реалізовано unit тести для PayloadExtractor (Foundry) та E2E тест для повного recovery flow. Тестування підтвердило працездатність cross-chain flow. Час відновлення становить ~3 хвилини (порівняно з 14 днями у Safe Candide та 48 годинами в Argent), gas costs становлять ~2.4M gas для setup та ~178K gas за recovery.

Практичне значення роботи полягає у створенні першої системи приватного соціального відновлення для Safe гаманців, що забезпечує повну приватність опікунів з використанням zero-knowledge proofs та підтримує крос-чейн розгортання.

Особистий внесок автора. У рамках кваліфікаційної роботи автором було самостійно виконано: дослідження та аналіз існуючих рішень соціального відновлення; проєктування крос-чейн архітектури системи; розробку смарт-контрактів на Solidity (SafeRecoveryModule, AztecRecoveryValidator) та Noir (RecoveryRegistry з приватним зберіганням опікунів та анонімним голосуванням); розробку Relayer-сервісу на Go для крос-чейн передачі повідомлень; розробку frontend на Next.js; розгортання та тестування на тестових мережах Sepolia та Aztec Devnet. Автор використовував існуючі технології як компоненти (Aztec Network, Safe Protocol, Wormhole Protocol, Foundry), при цьому вся логіка взаємодії між компонентами та інтеграції розроблені автором самостійно.

Подальший розвиток системи може включати: оптимізацію gas costs через batching транзакцій, підтримку інших типів гаманців (не лише Safe), реалізацію timelock-механізму для додаткової безпеки та інтеграцію з production-версією Aztec Network.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Antonopoulos A. M. Mastering Bitcoin: Programming the Open Blockchain. 2nd ed. O'Reilly Media, 2017. URL: https://github.com/bitcoinbook/bitcoinbook (дата звернення: 10.03.2026).
2. How Many Bitcoin Are Lost? Ledger Academy. URL: https://www.ledger.com/academy/topics/economics-and-regulation/how-many-bitcoin-are-lost-ledger (дата звернення: 15.02.2026).
3. Crypto Hacking Stolen Funds 2025 Report. Chainalysis, 2025. URL: https://www.chainalysis.com/blog/crypto-hacking-stolen-funds-2025/ (дата звернення: 20.02.2026).
4. Buterin V. Why we need wide adoption of social recovery wallets. 2021. URL: https://vitalik.eth.limo/general/2021/01/11/recovery.html (дата звернення: 10.02.2026).
5. Aztec Safe Recovery: GitHub Repository. URL: https://github.com/alik-eth/aztec-safe-recovery (дата звернення: 01.05.2026).
6. EIP-7579: Minimal Modular Smart Accounts. Ethereum Improvement Proposals. URL: https://eips.ethereum.org/EIPS/eip-7579 (дата звернення: 15.02.2026).
7. Safe Smart Account Modules Documentation. Safe Global. URL: https://docs.safe.global/advanced/smart-account-modules (дата звернення: 20.02.2026).
8. Aztec Protocol Documentation. URL: https://docs.aztec.network/ (дата звернення: 01.03.2026).
9. Noir Language Documentation. URL: https://noir-lang.org/docs/ (дата звернення: 01.03.2026).
10. Wormhole Documentation. URL: https://docs.wormhole.com/ (дата звернення: 01.03.2026).
11. Cross-Chain Catalysts Proposal: Deployment Grant AztecGuardianRecovery. Aztec Forum. URL: https://forum.aztec.network/t/cross-chain-catalysts-

proposal-deployment-grant-aztecguardianrecovery/8019 (дата звернення: 15.03.2026).

12. EIP-4337: Account Abstraction Using Alt Mempool. Ethereum Improvement Proposals. URL: https://eips.ethereum.org/EIPS/eip-4337 (дата звернення: 15.02.2026).
13. Goldwasser S., Micali S., Rackoff C. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*. 1989. Vol. 18, No. 1. P. 186–208.
14. Safe Protocol Documentation. URL: https://docs.safe.global/ (дата звернення: 20.02.2026).
15. Crypto Hacks Surged by 40% in 2024: Over $2.3 Billion Stolen. BeInCrypto. URL: https://beincrypto.com/crypto-hacks-2024-losses-reach-over-2-billion/ (дата звернення: 20.02.2026).
16. How to recover my wallet with guardians (onchain): complete guide. Argent Support. URL: https://support.argent.xyz/hc/en-us/articles/360007338877 (дата звернення: 25.02.2026).
17. Guardians. Loopring Smart Wallet Documentation. URL: https://docs-wallet.loopring.io/security/guardians (дата звернення: 25.02.2026).
18. Introducing Candide’s Social Recovery Module: Enabling Composable Recovery for Safe Smart Account. Safe Foundation. URL: https://safefoundation.org/blog/introducing-candides-social-recovery (дата звернення: 25.02.2026).
19. Safe Social Recovery Module Security Review. Ackee Blockchain, 2024. URL: https://ackee.xyz/blog/safe-social-recovery-module/ (дата звернення: 25.02.2026).
20. Social Recovery Wallet: GitHub Repository. URL: https://github.com/verumlotus/social-recovery-wallet (дата звернення: 01.03.2026).
21. Kethepalli Y. et al. Reinforcing Security and Usability of Crypto-Wallet with Post-Quantum Cryptography and Zero-Knowledge Proof. *arXiv preprint*. 2023. URL: https://arxiv.org/abs/2308.07309 (дата звернення: 10.03.2026).

22. A Survey on the Applications of Zero-Knowledge Proofs. *arXiv preprint*. 2024. URL: https://arxiv.org/abs/2408.00243 (дата звернення: 10.03.2026).
23. Leveraging Zero Knowledge Proofs for Blockchain-Based Identity Sharing: A Survey. *Computer Communications*. 2024. URL: https://www.sciencedirect.com/science/article/pii/S2214212623002624 (дата звернення: 10.03.2026).
24. Accounts. Aztec Documentation. URL: https://docs.aztec.network/aztec/concepts/accounts (дата звернення: 01.03.2026).
25. VAAs (Verified Action Approvals). Wormhole Documentation. URL: https://wormhole.com/docs/protocol/infrastructure/vaas/ (дата звернення: 01.03.2026).
26. Introducing Noir: The Universal Language of Zero-Knowledge. Aztec Labs Blog. URL: https://medium.com/aztec-protocol/introducing-noir-the-universal-language-of-zero-knowledge-ff43f38d86d9 (дата звернення: 01.03.2026).
27. Notes (UTXOs). Aztec Documentation. URL: https://docs.aztec.network/developers/docs/foundational-topics/state_management (дата звернення: 01.03.2026).
28. Keys. Aztec Documentation. URL: https://docs.aztec.network/aztec/concepts/accounts/keys (дата звернення: 01.03.2026).
29. Wormhole 101: Guardians. Wormhole Blog. URL: https://wormhole.com/blog/wormhole-101-guardians (дата звернення: 01.03.2026).
30. Aztec: The Hybrid zkRollup. Aztec Documentation. URL: https://docs.aztec.network/ (дата звернення: 15.03.2026).
31. Zero-knowledge proof. Wikipedia. URL: https://en.wikipedia.org/wiki/Zero-knowledge_proof (дата звернення: 10.03.2026).
32. 12 Solidity Smart Contract Security Best Practices. Alchemy. URL: https://www.alchemy.com/overviews/smart-contract-security-best-practices (дата звернення: 15.03.2026).

33. Wormhole Bridge Exploit Incident Analysis. CertiK. URL: https://www.certik.com/resources/blog/1kDYgyBcisoD2EqiBpHE5l-wormhole-bridge-exploit-incident-analysis (дата звернення: 15.03.2026).
34. Security. Wormhole Documentation. URL: https://wormhole.com/docs/protocol/security/ (дата звернення: 01.03.2026).
35. Zero-Knowledge Proof Vulnerability Analysis and Security Auditing. *IACR ePrint*. 2024. URL: https://eprint.iacr.org/2024/514.pdf (дата звернення: 20.03.2026).
36. Testing Contracts in the TXE. Aztec Documentation. URL: https://docs.aztec.network/guides/developer_guides/smart_contracts/testing_contracts/testing (дата звернення: 01.04.2026).
37. Fuzz Testing. Foundry Book. URL: https://getfoundry.sh/forge/fuzz-testing (дата звернення: 01.04.2026).
38. Run Aztec Sandbox. Aztec Documentation. URL: https://docs.aztec.network/developers/docs/guides/local_env/sandbox (дата звернення: 01.04.2026).
39. execTransactionFromModule. Safe Documentation. URL: https://docs.safe.global/reference-smart-account/modules/execTransactionFromModule (дата звернення: 20.02.2026).
40. addOwnerWithThreshold. Safe Documentation. URL: https://docs.safe.global/reference-smart-account/owners/addOwnerWithThreshold (дата звернення: 20.02.2026).
41. Man who threw away $180m in bitcoin hopes AI and robot dogs will find it. The Guardian, 2024. URL: https://www.theguardian.com/technology/2024/oct/09/james-howells-thrown-away-bitcoin-newport-landfill-wales (дата звернення: 10.03.2026).
42. Lost Passwords Lock Millionaires Out of Their Bitcoin Fortunes. The New York Times, 2021. URL: https://www.nytimes.com/2021/01/12/technology/bitcoin-passwords-wallets-fortunes.html (дата звернення: 10.03.2026).

ДОДАТОК А. СМАРТ-КОНТРАКТИ НА SOLIDITY

A.1 SafeRecoveryModule.sol

```
contract SafeRecoveryModule {
    // Immutable конфігурація
    IValidator7579 public immutable validator;
    IWormhole public immutable wormhole;

    // Mutable стан
    address public owner;
    bool public paused;

    // Safe → Aztec contract mapping (32-byte Aztec address)
    mapping(address => bytes32) public aztecRecoveryContract;

    // Replay protection
    mapping(bytes32 => bool) public consumedVaas;

    event RecoveryApplied(
        address indexed safe, uint256 threshold, bytes32 nonce
    );
    event AztecRecoveryContractSet(
        address indexed safe, bytes32 aztecContract
    );

    /// @notice Реєстрація Aztec контракту для Safe
    function setAztecRecoveryContract(bytes32 aztecContract)
external {
        aztecRecoveryContract[msg.sender] = aztecContract;
        emit AztecRecoveryContractSet(msg.sender, aztecContract);
    }

    /// @notice Виконання відновлення через VAA
    function verify(bytes calldata vaa) external {
        require(!paused, "ModulePaused");
        (IWormhole.VM memory vm, bool valid, ) =
            wormhole.parseAndVerifyVM(vaa);
        require(valid, "InvalidVaa");
        require(!consumedVaas[vm.hash], "VaaAlreadyConsumed");
        consumedVaas[vm.hash] = true;
        (
```

```
            address safeAddress,
            address newOwner,
            uint256 threshold
        ) = _decodePayload(vm.payload);
        require(
            aztecRecoveryContract[safeAddress] ==
vm.emitterAddress,
            "WrongEmitter"
        );
        _addOwner(safeAddress, newOwner, threshold);
    }

    function _moduleCall(
        address safe, address to, bytes memory data
    ) internal returns (bool success) {
        success = ISafe(safe).execTransactionFromModule(
            to, 0, data, Enum.Operation.Call
        );
    }

    function _addOwner(
        address safe, address newOwner, uint256 threshold
    ) internal {
        bytes memory data = abi.encodeWithSelector(
            ISafe.addOwnerWithThreshold.selector,
            newOwner, threshold
        );
        require(_moduleCall(safe, safe, data), "SafeCallFailed");
    }
}
```

A.2 AztecRecoveryValidator.sol

```
contract AztecRecoveryValidator is IValidator7579 {
    IWormhole public immutable wormhole;
    address public owner;
    bool public paused;

    mapping(address => bool) public moduleAuthorized;
    mapping(bytes32 => bool) public consumed;

    struct PayloadData {
```

```
        uint8 version;
        uint256 chainId;
        address safeAddress;
        address[] newOwners;
        uint256 threshold;
        bytes32 nonce;
        uint256 expiry;
    }

    function validate(
        bytes calldata vaa, bytes calldata callData
    ) external returns (bool) {
        require(!paused, "Paused");
        require(moduleAuthorized[msg.sender], "Unauthorized");
        (IWormhole.VM memory vm, bool valid, ) =
            wormhole.parseAndVerifyVM(vaa);
        require(valid, "InvalidVaa");
        PayloadData memory payload = _decodePayload(vm.payload);
        require(payload.version == 1, "WrongVersion");
        require(payload.chainId == block.chainid, "WrongChain");
        require(block.timestamp <= payload.expiry, "Expired");
        require(!consumed[payload.nonce], "NonceConsumed");
        require(
            _validateCallData(callData, payload),
            "CallDataMismatch"
        );
        return true;
    }
}
```

ДОДАТОК Б. AZTEC КОНТРАКТ (NOIR)

Б.1 Notes (notes.nr)

```
#[note]
pub struct GuardianNote {
    guardian: AztecAddress,
    safe_address: Field,
    owner: AztecAddress,
    randomness: Field,
}

impl GuardianNote {
    pub fn new(
        guardian: AztecAddress,
        safe_address: Field,
        owner: AztecAddress
    ) -> Self {
        GuardianNote {
            guardian, safe_address, owner,
            randomness: unsafe { random() },
        }
    }
}

#[note]
pub struct VoteNote {
    votes: Field,
    finished: bool,
    safe_address: Field,
    owner: AztecAddress,
    randomness: Field,
}

#[note]
pub struct GuardianRecordNote {
    guardian: AztecAddress,
    alias: Field,
    owner: AztecAddress,
    randomness: Field,
}
```

Б.2 RecoveryRegistry (main.nr)

```
contract RecoveryRegistry {
    use dep::aztec::prelude::*;
    use crate::notes::{GuardianNote, VoteNote, GuardianRecordNote};

    #[storage]
    struct Storage<Context> {
        config: PublicImmutable<Config, Context>,
        threshold: SharedMutable<u32, 180, Context>,
        guardians: PrivateSet<AddressNote, Context>,
        guardian_registered: Map<AztecAddress,
            PublicMutable<bool>, Context>,
        guardian_count: PublicMutable<u32, Context>,
        vote_counts: Map<AztecAddress, PublicMutable<u32>,
        Context>,
        recovery_sent: Map<AztecAddress, PublicMutable<bool>,
        Context>,
        guardian_records: PrivateSet<GuardianRecordNote, Context>,
    }

    #[public]
    #[initializer]
    fn constructor(
        owner: AztecAddress,
        wormhole_address: AztecAddress,
        chain_id: Field,
        safe_address: Field
    ) {
        let config = Config {
            owner, wormhole_address, chain_id, safe_address,
        };
        storage.config.initialize(config);
        storage.threshold.schedule_value_change(1);
        storage.guardian_count.write(0);
    }

    #[private]
    fn add_guardian(guardian: AztecAddress, alias: Field) {
        let owner = storage.config.read().owner;
        assert(context.msg_sender() == owner, "Only owner");
        let record = GuardianRecordNote::new(guardian, alias,
        owner);
        storage.guardian_records.insert(&mut record);
```

```
        let guardian_note = AddressNote::new(guardian, guardian);
        storage.guardians.insert(&mut guardian_note);
        RecoveryRegistry::at(context.this_address())
            .register_guardian_public(guardian)
            .enqueue(&mut context);
    }

    #[public]
    #[internal]
    fn register_guardian_public(guardian: AztecAddress) {
        storage.guardian_registered.at(guardian).write(true);
        let count = storage.guardian_count.read();
        storage.guardian_count.write(count + 1);
        let new_threshold = (count + 1) / 2 + 1;
        storage.threshold.schedule_value_change(new_threshold);
    }

    #[private]
    fn vote(candidate: AztecAddress) {
        let sender = context.msg_sender();
        let guardian_note = storage.guardians.get_note(sender);
        assert(guardian_note.get_address() == sender, "Not
        guardian");
        let nullifier = std::hash::pedersen([
            sender.to_field(),
            candidate.to_field(),
            storage.config.read().safe_address
        ]);
        context.push_nullifier(nullifier[0]);
        RecoveryRegistry::at(context.this_address())
            .increment_vote_internal(candidate)
            .enqueue(&mut context);
    }

    #[public]
    #[internal]
    fn increment_vote_internal(candidate: AztecAddress) {
        let current = storage.vote_counts.at(candidate).read();
        let new_count = current + 1;
        storage.vote_counts.at(candidate).write(new_count);
        let threshold = storage.threshold.read();
        if new_count >= threshold {
            let already_sent =
                storage.recovery_sent.at(candidate).read();
```

```
            if !already_sent {
                storage.recovery_sent.at(candidate).write(true);
                _send_wormhole_message(candidate);
            }
        }
    }

    fn _send_wormhole_message(candidate: AztecAddress) {
        let config = storage.config.read();
        let payload = RecoveryPayload {
            version: 1,
            chain_id: config.chain_id,
            safe_address: config.safe_address,
            new_owner: candidate.to_field(),
            threshold: storage.threshold.read(),
            nonce: std::hash::pedersen([
                config.safe_address,
                candidate.to_field(),
                context.block_number()
            ])[0],
        };
        let wormhole = Wormhole::at(config.wormhole_address);
        wormhole.publish_message(payload.serialize())
            .call(&mut context);
    }
}
```

ДОДАТОК В. RELAYER SERVICE (GO)

```go
// relayer.go
type SpyClient struct {
    endpoint string
    conn     *grpc.ClientConn
    client   spy.SpyRPCServiceClient
}

func (s *SpyClient) Connect() error {
    var err error
    for attempt := 0; attempt < 5; attempt++ {
        s.conn, err = grpc.Dial(s.endpoint,

        grpc.WithTransportCredentials(insecure.NewCredentials()))
        if err == nil {
            s.client = spy.NewSpyRPCServiceClient(s.conn)
            return nil
        }
        time.Sleep(2 * time.Second)
    }
    return fmt.Errorf("failed to connect after 5 attempts: %w",
        err)
}

func (s *SpyClient) Subscribe(ctx context.Context) (
    spy.SpyRPCService_SubscribeSignedVAAClient, error) {
    req := &spy.SubscribeSignedVAARequest{
        Filters: []*spy.FilterEntry{{
            Filter: &spy.FilterEntry_EmitterFilter{
                EmitterFilter: &spy.EmitterFilter{
                    ChainId: uint32(sourceChainId),
                },
            },
        }},
    }
    return s.client.SubscribeSignedVAA(ctx, req)
}

type Deduplicator struct {
    seen map[string]time.Time
    ttl  time.Duration
    mu   sync.RWMutex
```

```
}

func (d *Deduplicator) IsDuplicate(vaa []byte) bool {
    hash := sha256.Sum256(vaa)
    key := hex.EncodeToString(hash[:])
    d.mu.RLock()
    if seen, ok := d.seen[key]; ok {
        if time.Since(seen) < d.ttl {
            d.mu.RUnlock()
            return true
        }
    }
    d.mu.RUnlock()
    d.mu.Lock()
    d.seen[key] = time.Now()
    d.mu.Unlock()
    return false
}

type EmitterRegistry struct {
    client   *ethclient.Client
    module   common.Address
    emitters map[string]common.Address
    fromBlock uint64
    mu       sync.RWMutex
}

func (r *EmitterRegistry) LoadHistorical() error {
    query := ethereum.FilterQuery{
        FromBlock: big.NewInt(int64(r.fromBlock)),
        Addresses: []common.Address{r.module},
        Topics: [][]common.Hash{{
            crypto.Keccak256Hash([]byte(
                "AztecRecoveryContractSet(address,bytes32)")),
        }},
    }
    logs, err := r.client.FilterLogs(context.Background(), query)
    if err != nil {
        return err
    }
    for _, log := range logs {
        safe := common.BytesToAddress(log.Topics[1].Bytes())
        aztec := common.BytesToHash(log.Data)
        r.mu.Lock()
```

```
        r.emitters[aztec.Hex()] = safe
        r.mu.Unlock()
    }
    return nil
}

type EVMClient struct {
    client     *ethclient.Client
    privateKey *ecdsa.PrivateKey
    module     common.Address
    chainId    *big.Int
}

func (e *EVMClient) SubmitVAA(vaa []byte) error {
    verifyABI, _ := abi.JSON(strings.NewReader(
        `[{"name":"verify","type":"function",` +
        `"inputs":[{"name":"vaa","type":"bytes"}]}]`))
    data, err := verifyABI.Pack("verify", vaa)
    if err != nil {
        return err
    }
    nonce, err := e.client.PendingNonceAt(
        context.Background(),
        crypto.PubkeyToAddress(e.privateKey.PublicKey))
    if err != nil {
        return err
    }
    gasPrice, _ := e.client.SuggestGasPrice(context.Background())
    tx := types.NewTransaction(
        nonce, e.module, big.NewInt(0),
        500000, gasPrice, data,
    )
    signedTx, _ := types.SignTx(tx,
        types.NewEIP155Signer(e.chainId), e.privateKey)
    return e.client.SendTransaction(context.Background(), signedTx)
}
```

ДОДАТОК Г. ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

Г.1 Підключення Safe App

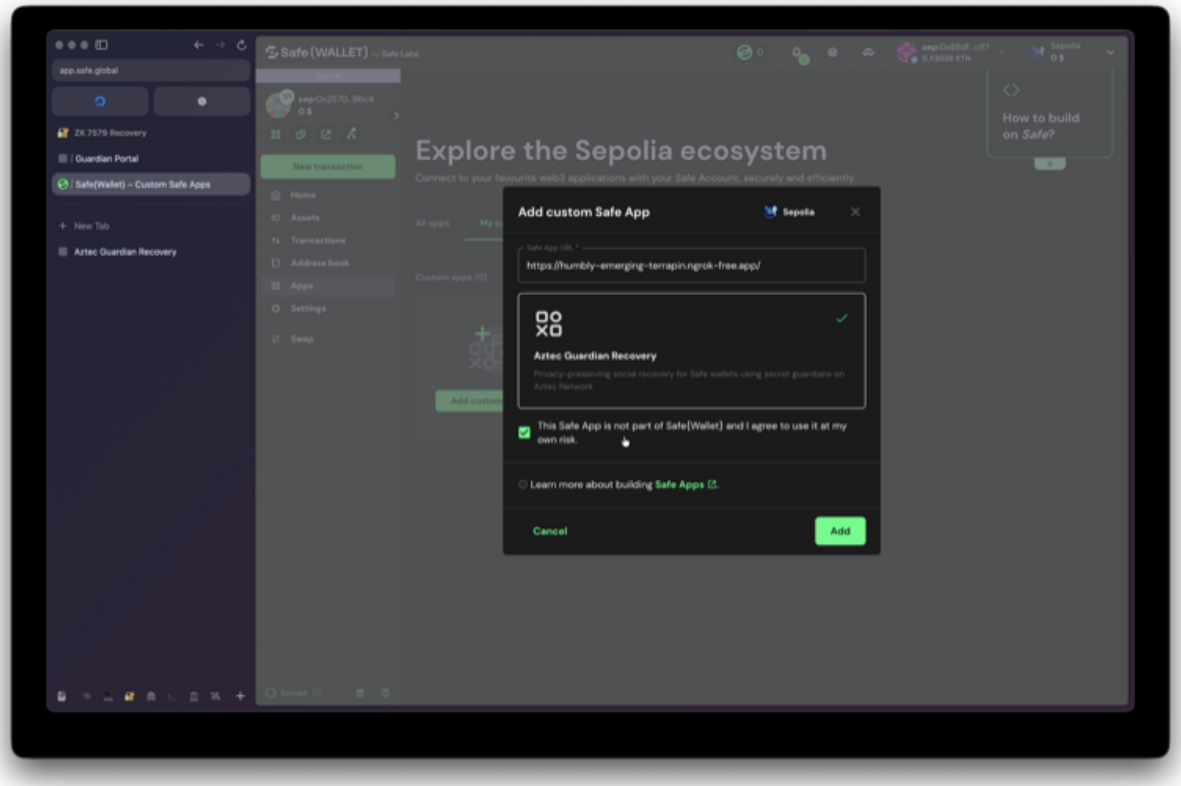

Рисунок Г.1 — Додавання Aztec Guardian Recovery як Custom Safe App

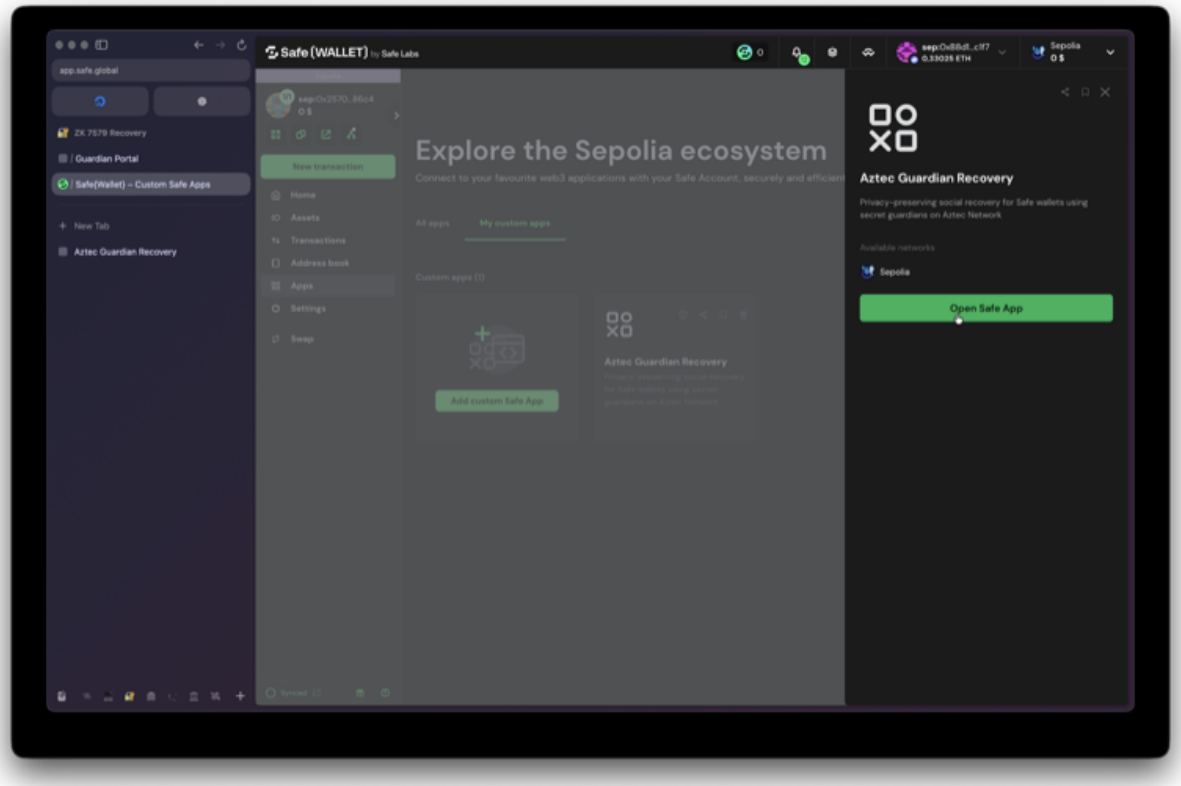

Рисунок Г.2 — Головна сторінка додатку після підключення

Г.2 Налаштування відновлення (Setup Wizard)

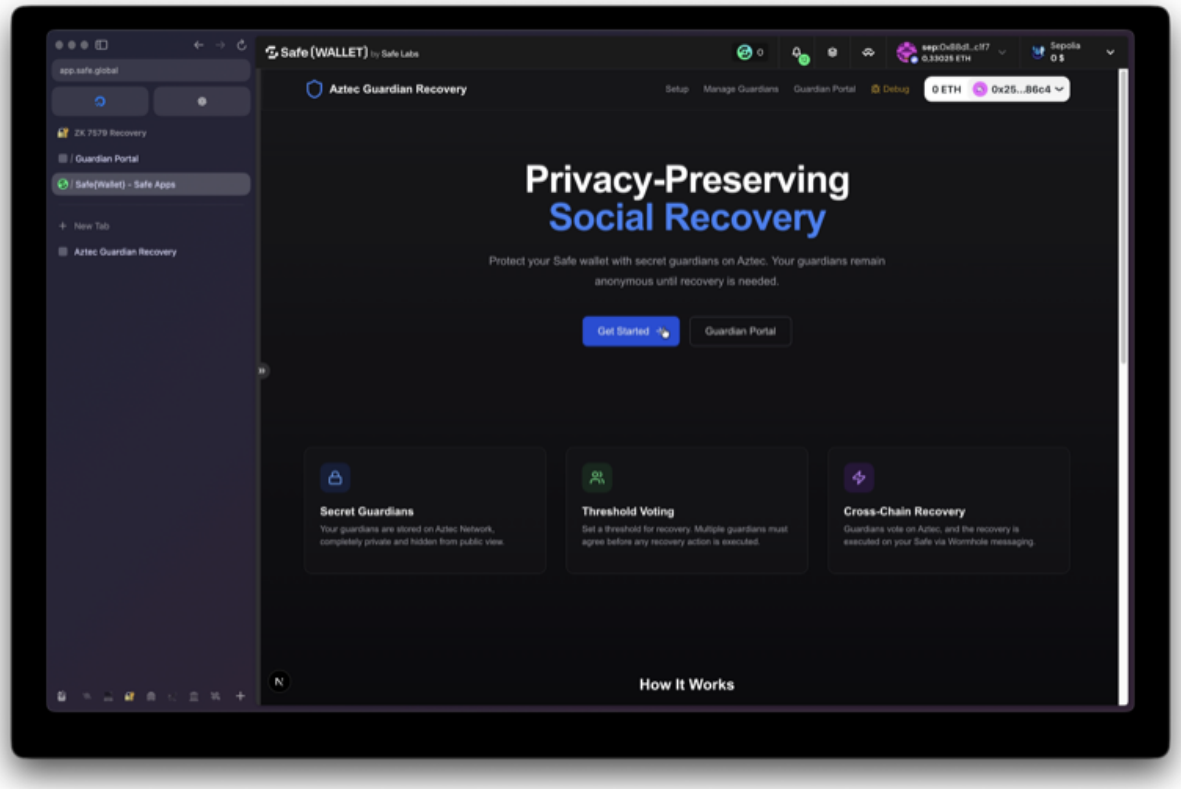

Рисунок Г.3 — Крок 1: Підключення Safe гаманця

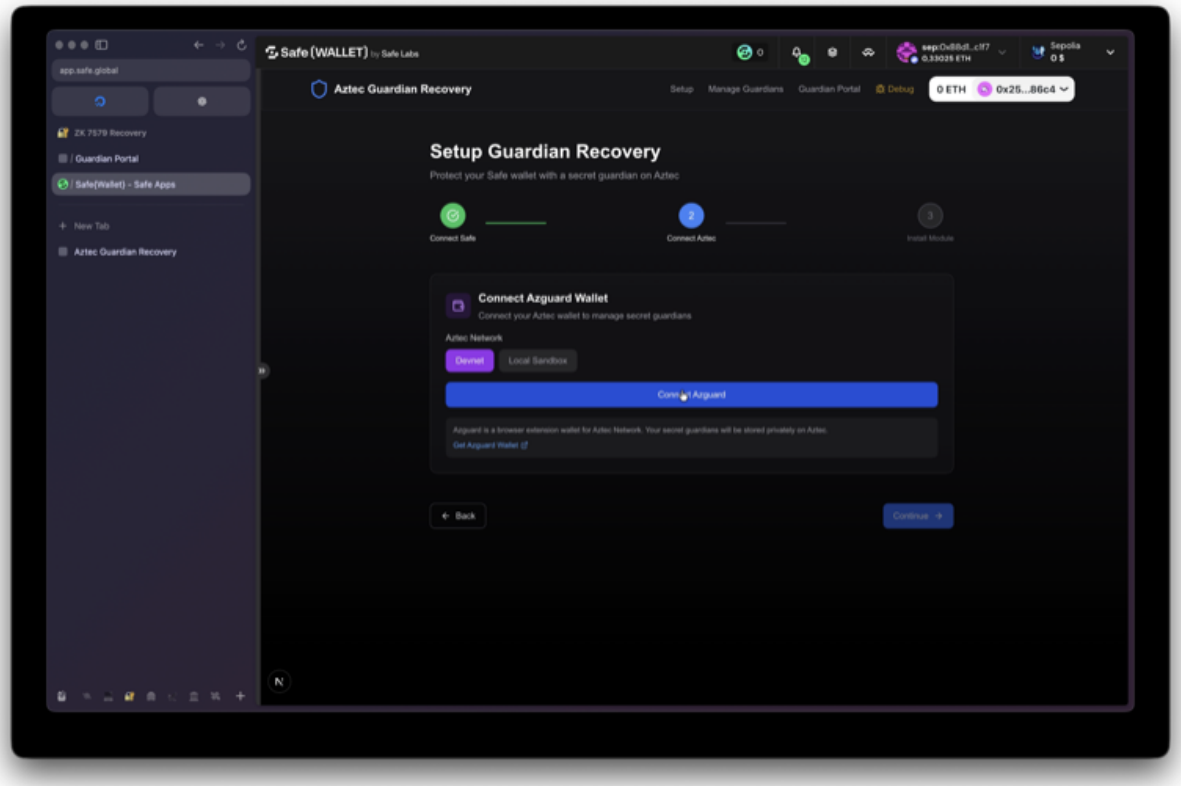

Рисунок Г.4 — Крок 2: Підключення Aztec гаманця

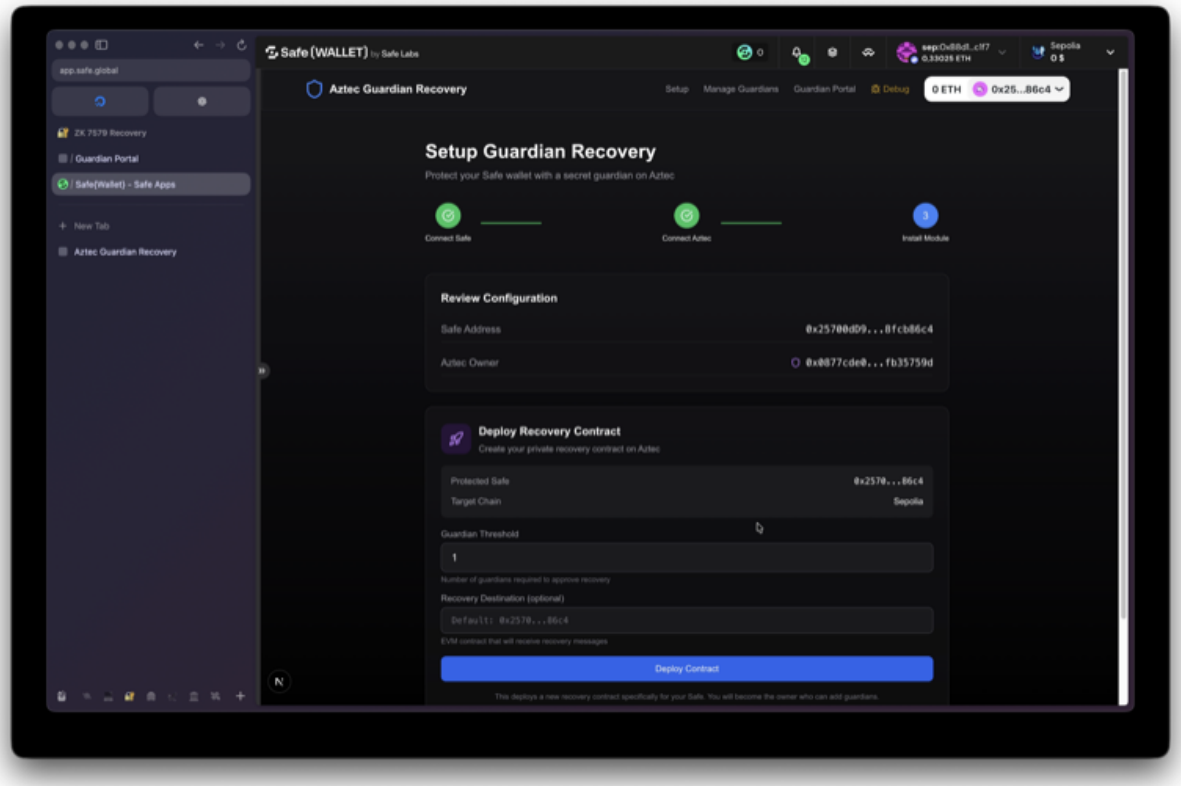

Рисунок Г.5 — Крок 3: Конфігурація та розгортання Recovery Contract на Aztec

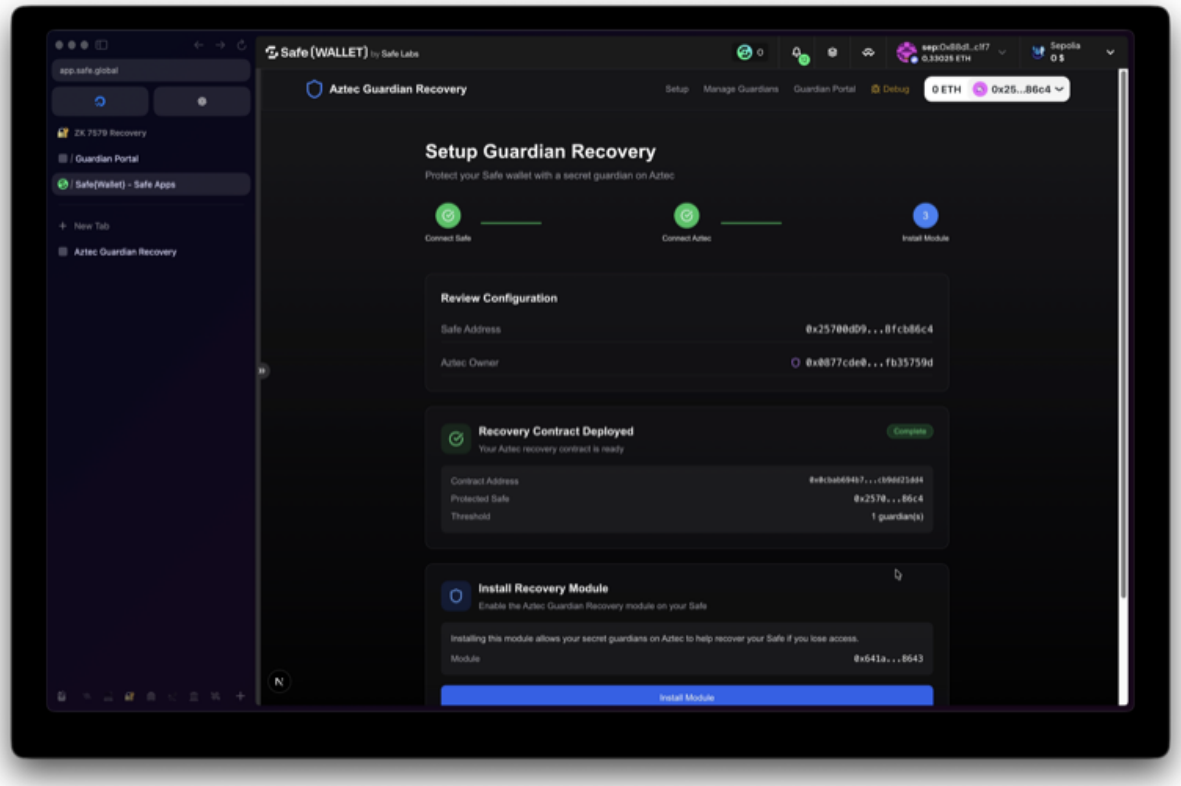

Рисунок Г.6 — Процес розгортання контракту на Aztec Network

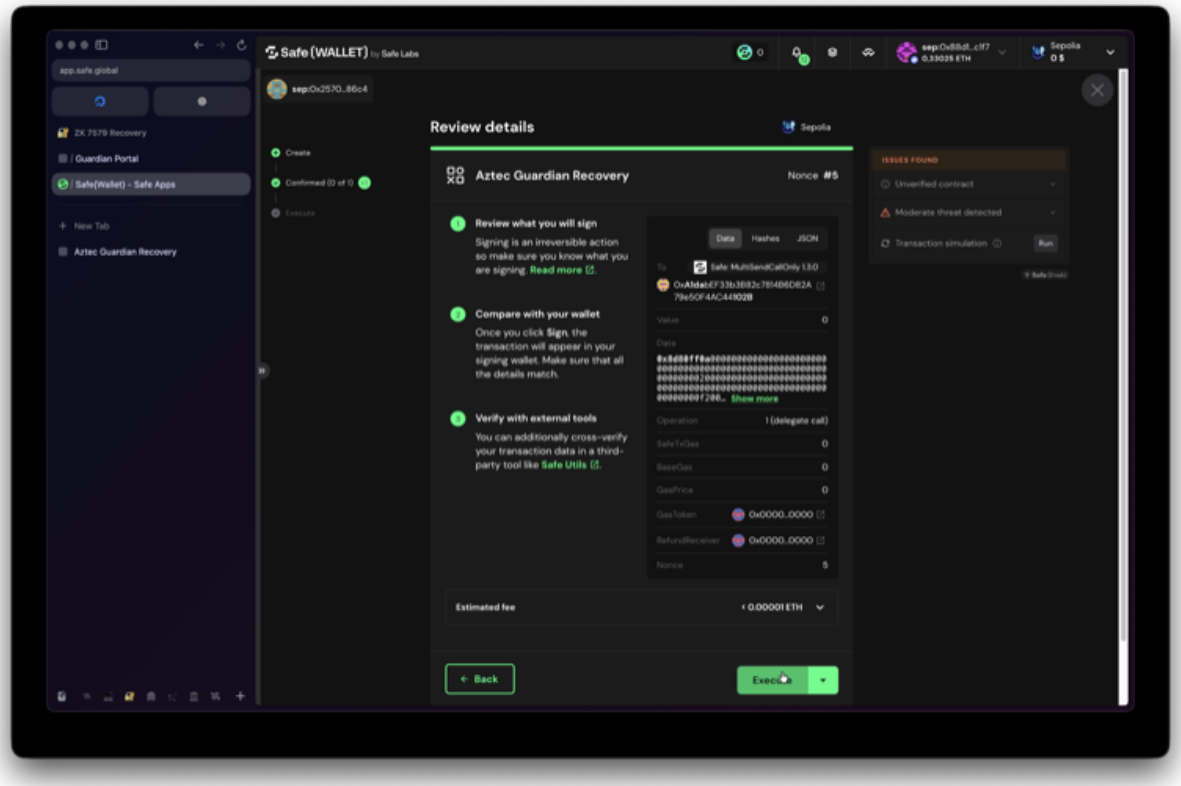

Рисунок Г.7 — Контракт успішно розгорнуто

Г.3 Встановлення модуля на Safe

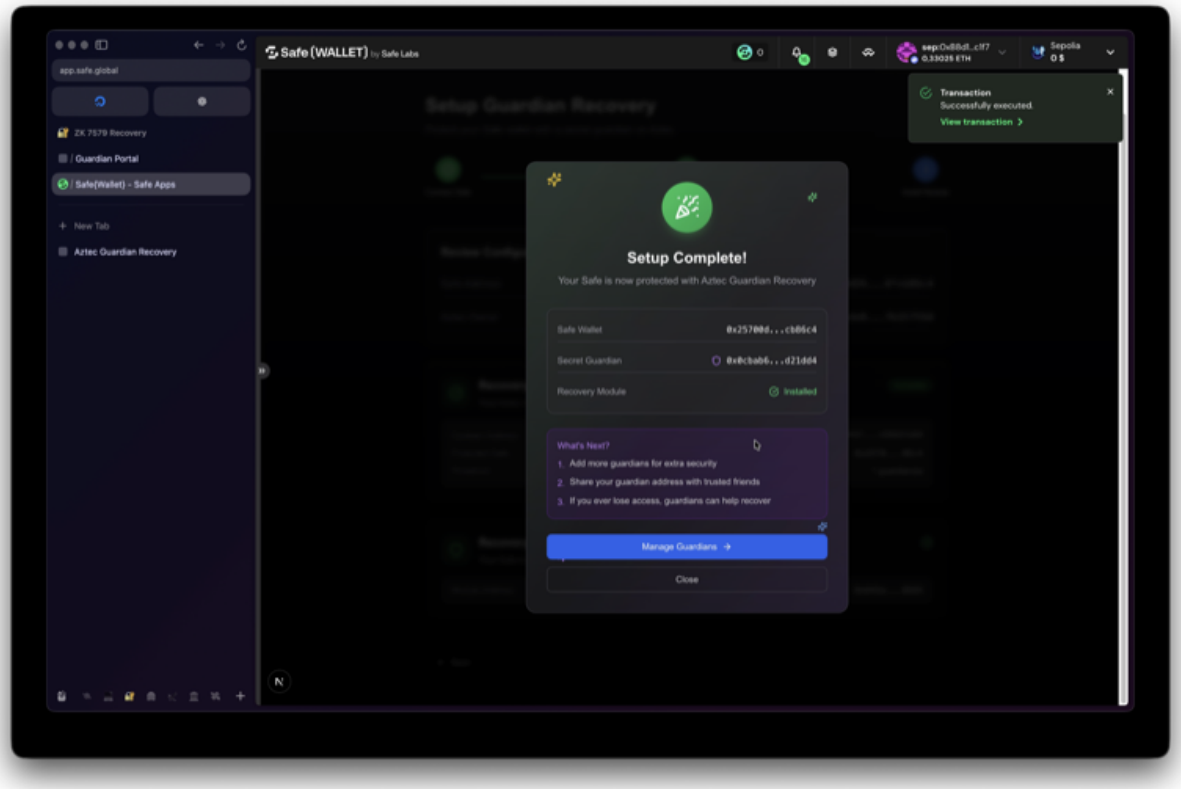

Рисунок Г.8 — Встановлення SafeRecoveryModule на Safe гаманець

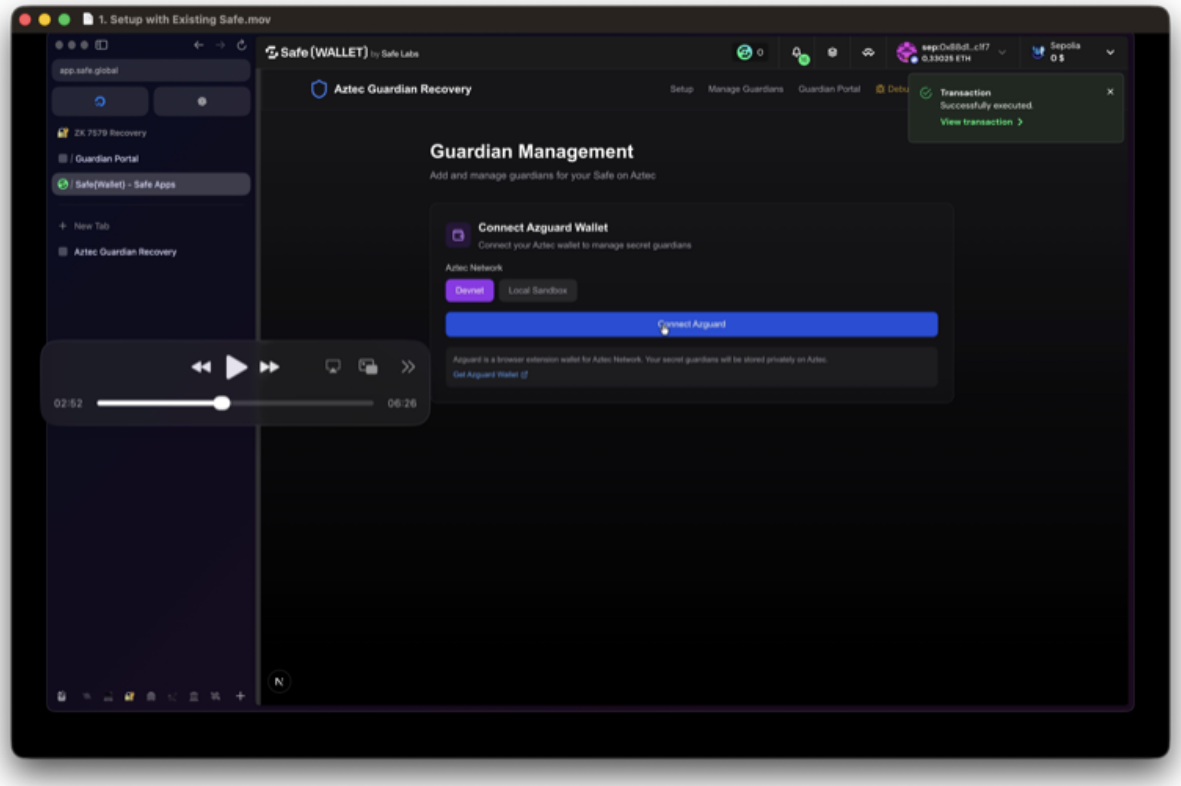

Рисунок Г.9 — Підтвердження транзакції встановлення модуля

Г.4 Управління опікунами

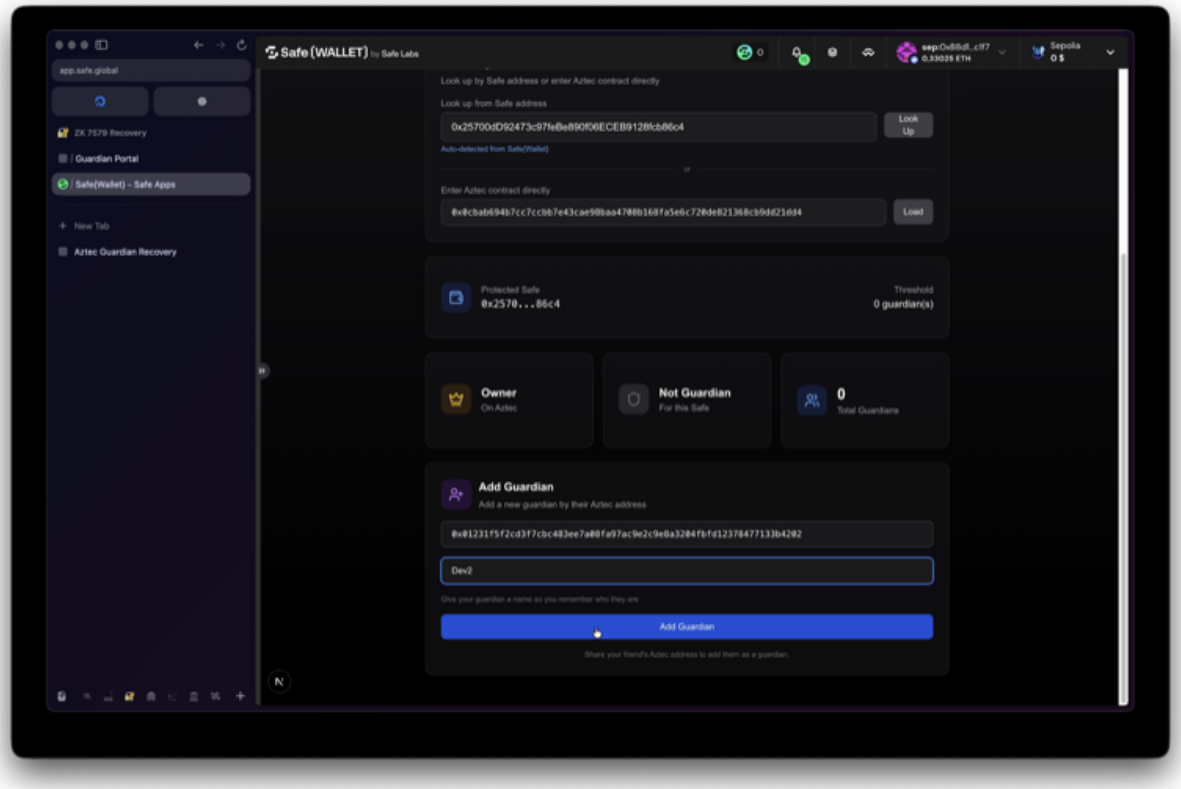

Рисунок Г.10 — Додавання опікуна за Aztec-адресою

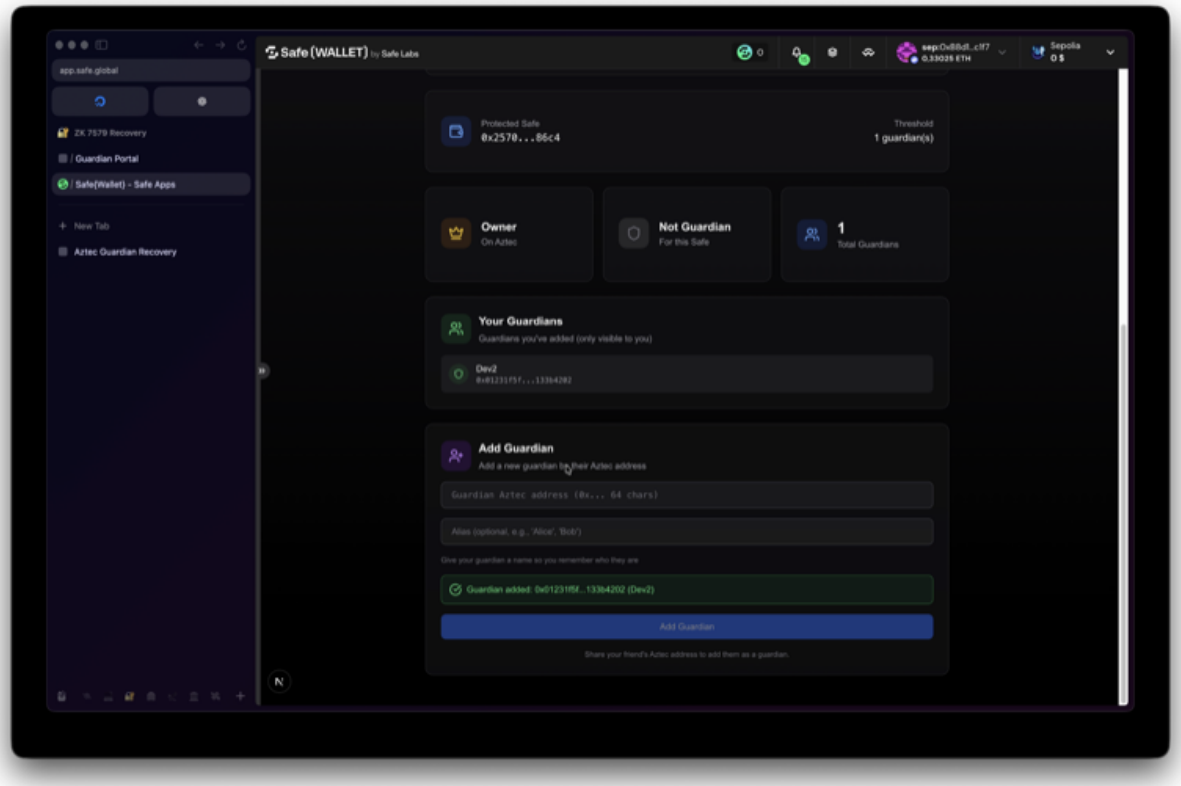

Рисунок Г.11 — Опікун успішно доданий (приватна транзакція на Aztec)

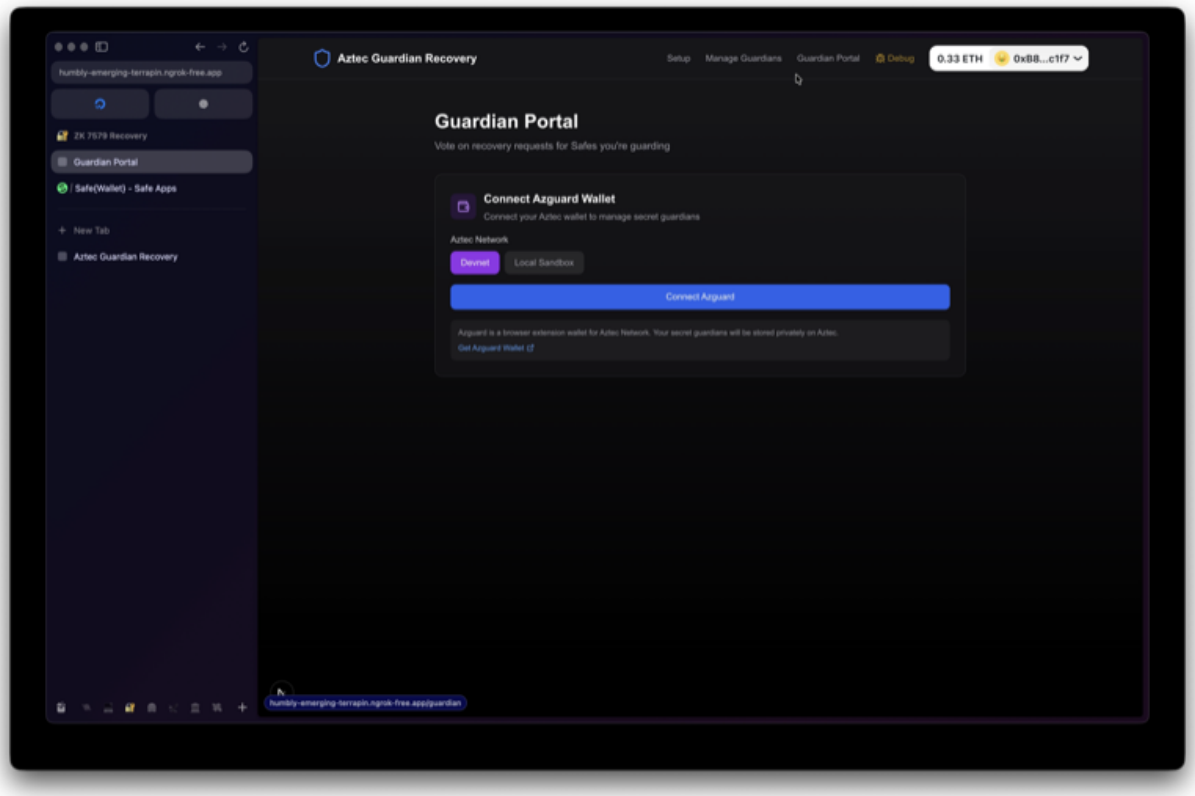

Рисунок Г.12 — Список зареєстрованих опікунів

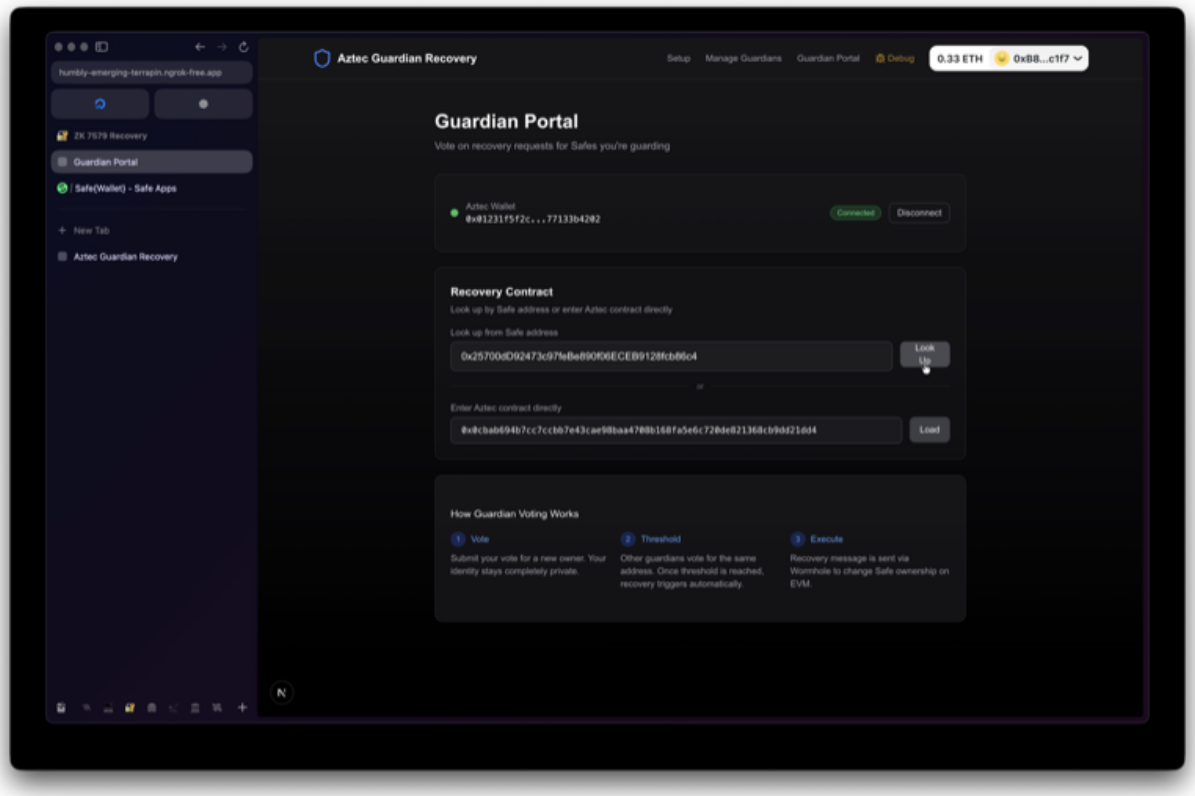

Рисунок Г.13 — Debug-панель з інформацією про контракт

Г.5 Голосування за відновлення (Guardian Portal)

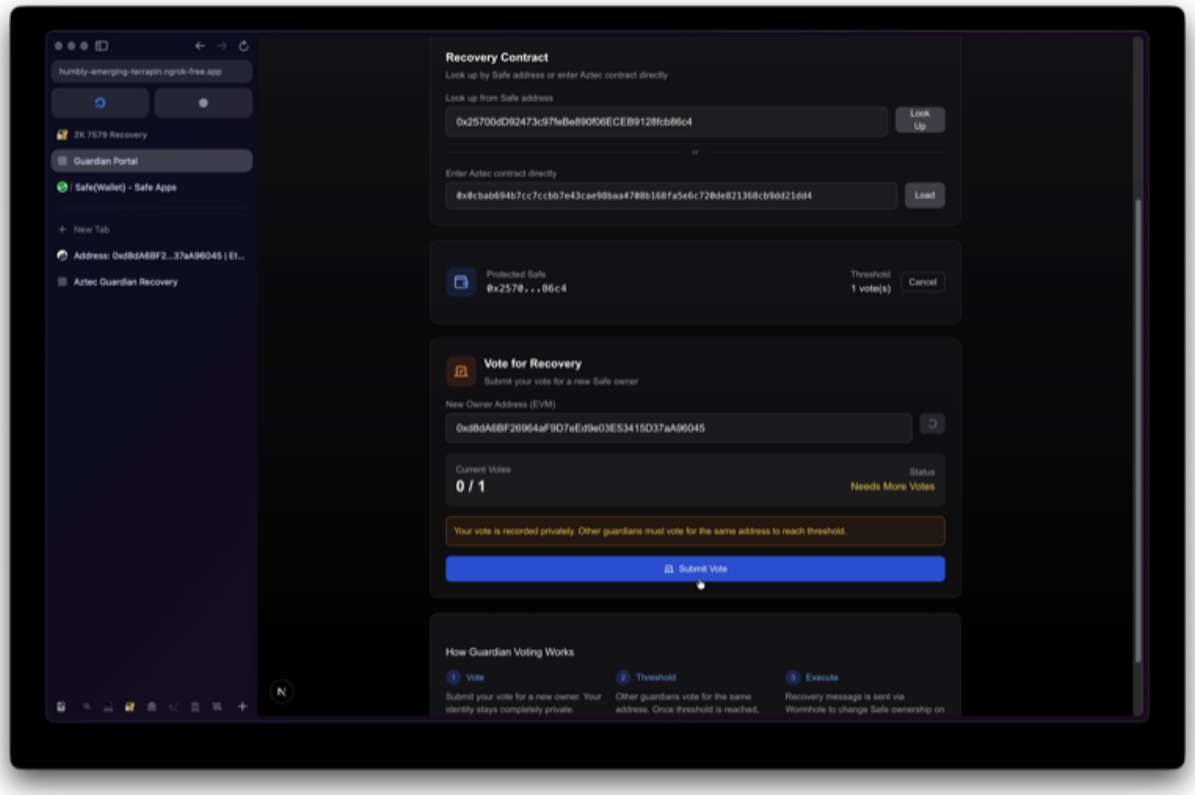

Рисунок Г.14 — Guardian Portal: введення адреси нового власника та голосування

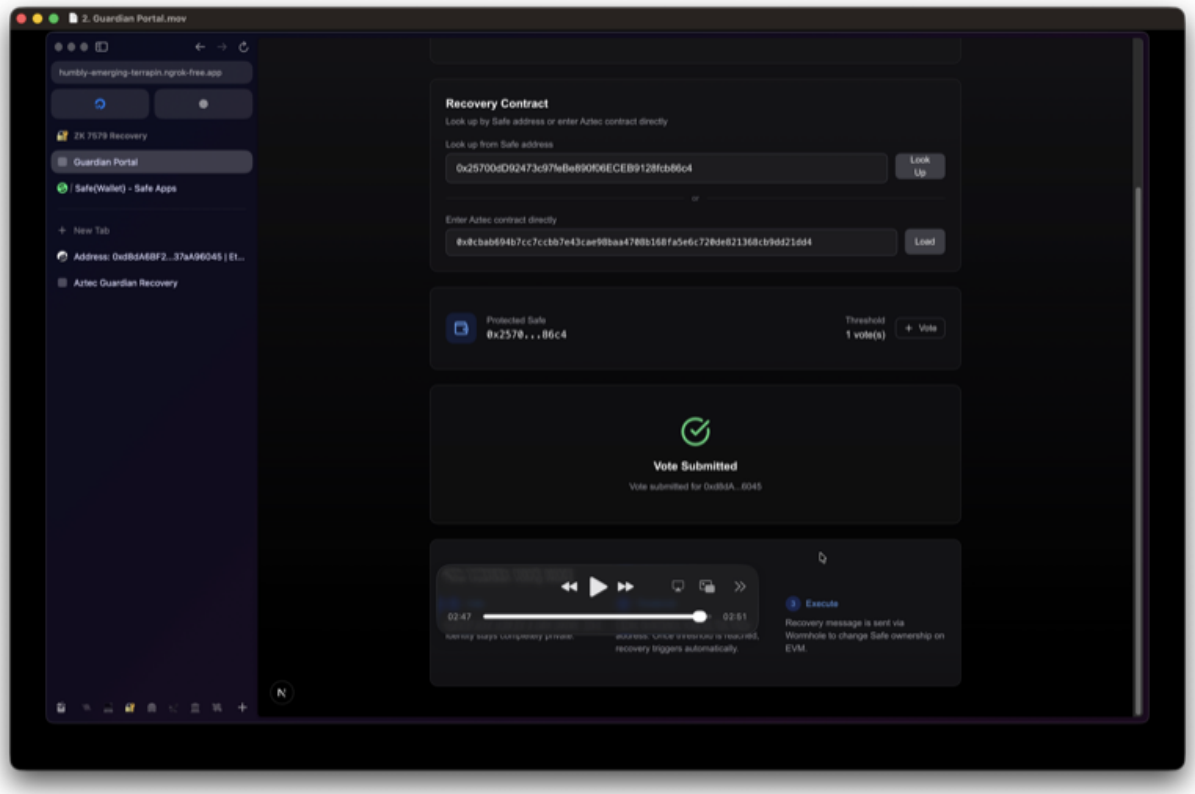

Рисунок Г.15 — Підтвердження голосу (приватна транзакція)

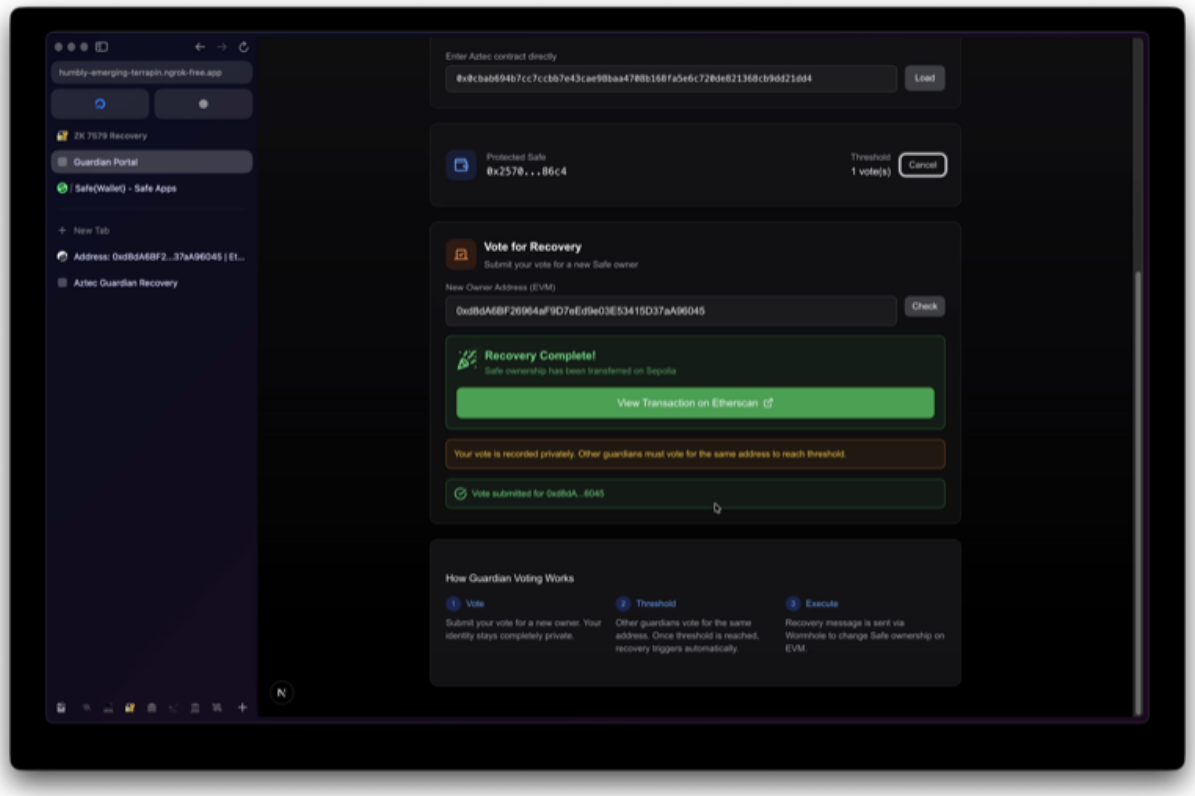

Рисунок Г.16 — Голос зараховано, очікування досягнення порогу

Г.6 Результат відновлення

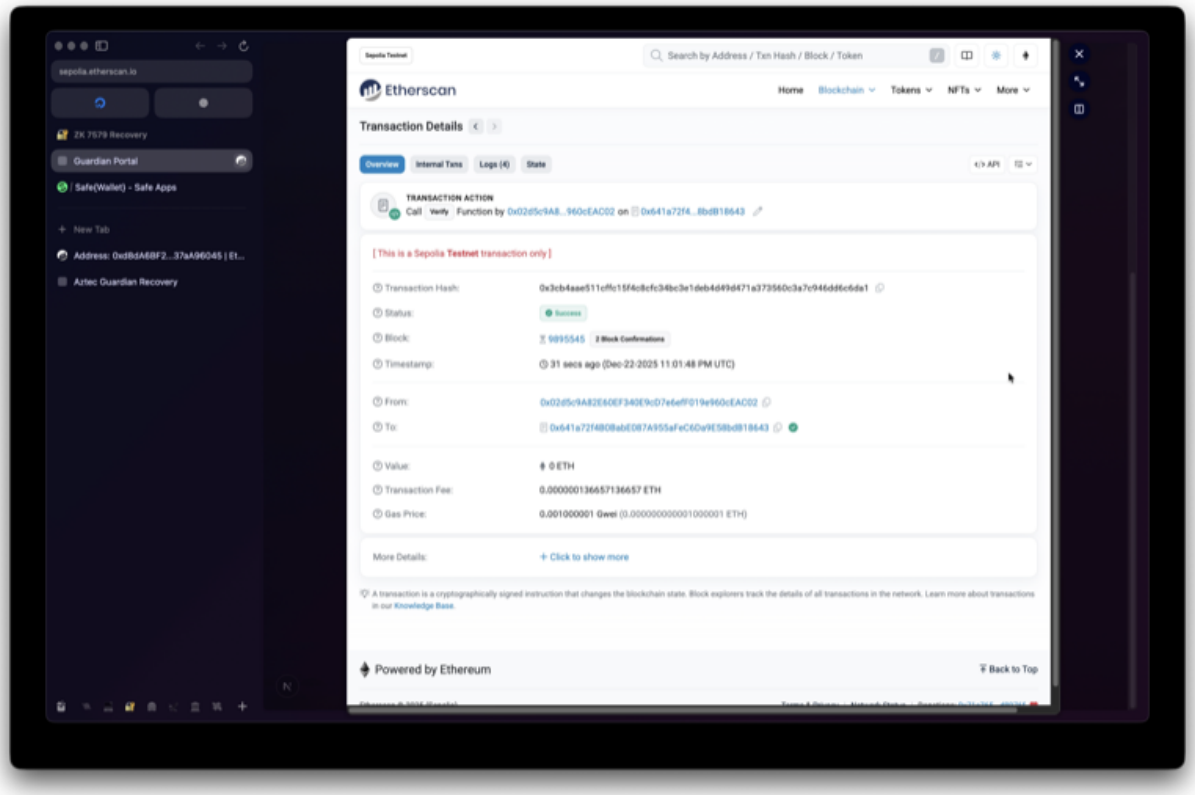

Рисунок Г.17 — Recovery виконано: Wormhole VAA доставлено на EVM

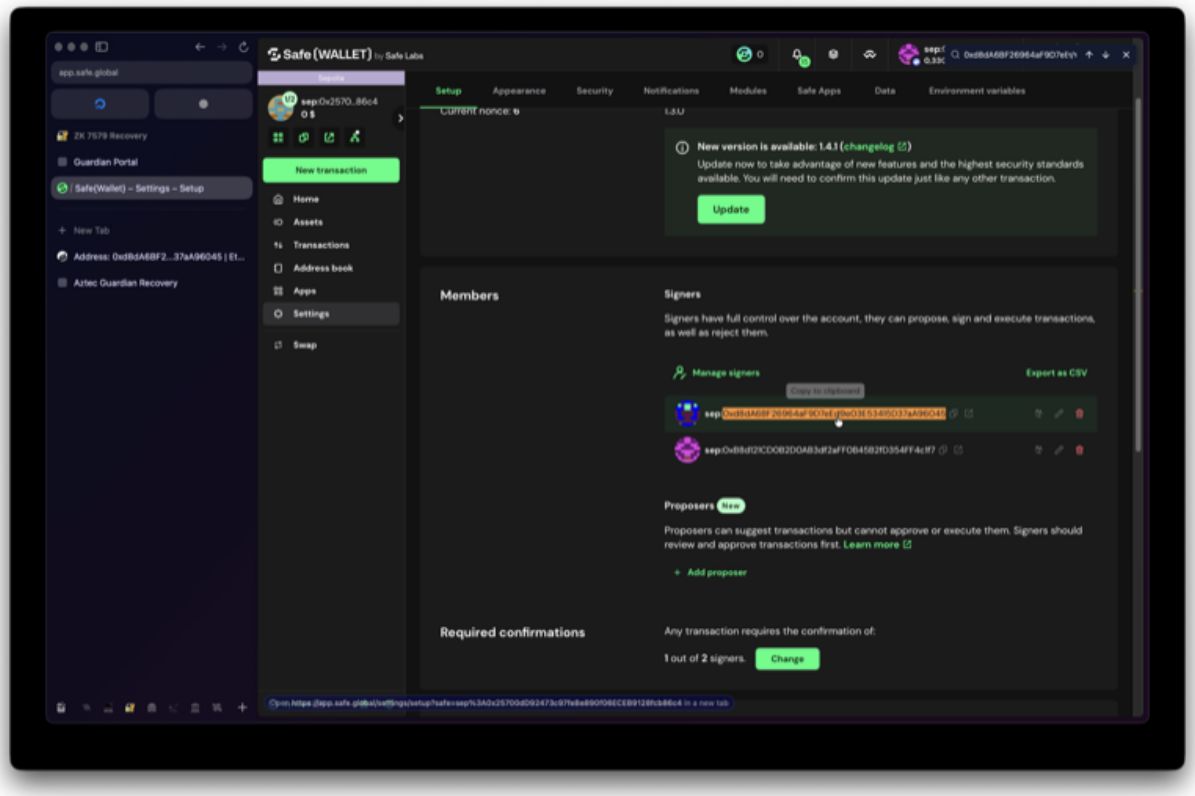

Рисунок Г.18 — Safe Settings: новий власник успішно додано до гаманця